The ActivePolygon Polygonal Algorithm for Haptic Force Generation

Tom Anderson
Nick Brown
Novint Technologies

Abstract

Algorithms for computing forces and associated surface deformations from a polygonal data set are given, which can be used to haptically and graphically display virtual objects with a single-point cursor. A culled collision detection algorithm is described that works in real-time with large data sets utilizing an oct-tree method. After a collision is detected, forces are created based on the local area near the cursor, keeping track of an active polygon. This creates a method that is effective and scalable for large models. The 'Bendable Polygon' algorithm for visual rendering of computer generated surfaces is also given.

Introduction

Many of the most common data formats for 3D computer-generated worlds utilize polygonal representations for objects. In order to take advantage of the large existing quantity of polygonal data sets, and a common standard in environments and hardware acceleration, a method for creating haptics forces based on polygons is necessary. The following paper describes an algorithm that can be used to create the forces on polygonal objects. The ActivePolygon algorithm was implemented with triangular datasets, however the concepts can apply to any type of polygonal dataset.

The algorithm first focuses on determining if the user point, or cursor, has touched an object, which requires a collision detection algorithm. A culled collision detection algorithm is described that works in real-time with large data sets. Then forces are created based on penetration depth and the relative position of the cursor to the object's facets. Graphically, the Bendable Polygon technique is described in which a local area of an object is broken up to allow for small-scale visual deformations. Larger scale deformations occur in the haptic domain through a system of springs and dampers.

Collision Detection

The first step in creating the forces for an object is to find if the cursor is touching the object. This means that as the cursor moves, collisions between the cursor and the object's facets must be checked. After a collision is detected, forces are then determined and presented to the user.

A simple way to do collision detection is to check if the cursor has moved through any of the polygons in an object. This can be accomplished by taking the line segment from the cursor's current and previous positions each loop of the cycle, and comparing that segment with every one of the polygons in an object. If the line segment intersects any of the polygons then a collision has occurred.

This can be extremely time consuming, however, if the object consists of many polygons. It is inefficient to check every one of the polygons in an object each cycle of the loop. The process can be sped up by pre-processing the data and culling the polygons that are not in the cursor's vicinity during

run-time, which allows real-time collision detection even with large data sets. An oct-tree is used to subdivide the space around the object. During collision detection, each cycle of the haptic loop all leaf nodes which the cursor has moved through are determined (usually this is only the current leaf node) and only the polygons within these nodes are checked for collisions.

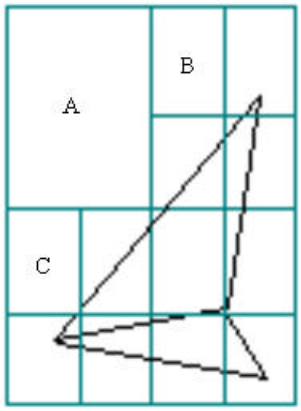


Figure 1: oct-tree based culling of a polygonal object.
Nodes A, B and C are empty, the remaining nodes contain one or both of the polygons.

To populate the oct-tree, first all the polygons are added to a single parent node. This node is then divided into eight octants all polygons within the parent are checked for overlapping each child octant, and added to the child's polygon list. Then, each of the octants is divided and it's polygon list transferred to the new children. An octant is not subdivided if contains a minimum number of polygons.

In pre-processing the oct-tree culling data, there are three situations in which a polygon should be included in an octant's checking domain as shown in Figure 2. The first is when any vertices of the polygon lie within the octant. The second occurs when any of the edges of the polygon intersect any of the sides of the octant's. The third situation occurs when any of the edges of the octant's intersect the polygon.

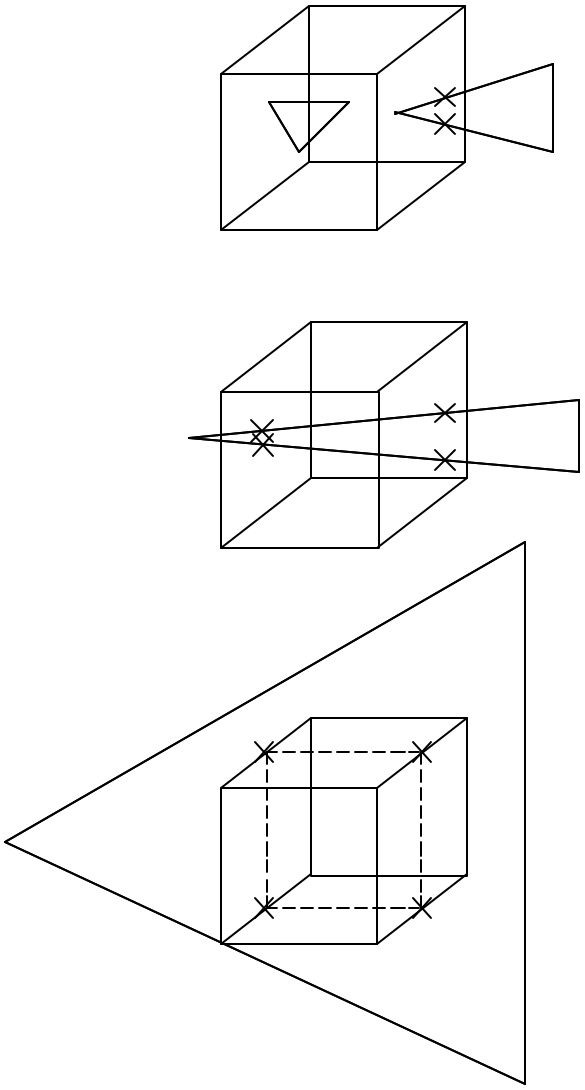


Figure 2: Voxel-Polygon events.
Top: 1 or more vertices in the voxel.
Middle: Polygon edge intersects a voxel side.
Bottom: Voxel edge intersects the polygon

If the tool's start position in the X direction is larger than its end position, octants are queried from right to left, otherwise from left to right. Setting the same checks in Y and Z ensures that the octant that will be collided with first is checked first.

The main consideration in using this type of culling comes from trade-offs in time and memory usage. The culling represents a method in which the object's size can grow to contain many millions of polygons, and the algorithm would still be able to do the collision detection in real-time. This would require very large amounts of memory, however. The maximum depth of the oct-tree and the maximum number of polygons within a leaf node can be changed to make a trade off between memory usage and processing time.

Force Generation

After a collision is detected, the forces must be presented. When the cursor touches an object, the specific polygon that was initially touched becomes the active polygon. The forces are in the normal direction and are proportional to the penetration depth into an object, which is measured from the currently active polygon. The direction of the force is interpolated at edges as the cursor moves across an object, and as the current polygon changes, to create a smooth feeling across facets. When the penetration depth of the active polygon becomes negative, the cursor has left the object, the forces are discontinued, and the collision detection algorithm is used again.

Normal Direction for Polygons

An initial issue is encountered because of the nature of a polygonal data set. The outward direction on a polygon is determined from the ordering of the points it contains. The direction that is considered outward is important for both graphics and haptics. In graphics, the outward direction is used to determine shading effects. In haptics, the outward direction is used in collision detection, force direction, and in interpolating between polygons.

A typical way to overcome this problem, which has become a standard in graphics, is to pre-process the points and order them so that all the vertex listings are consistent. If a data set is not already vertex-ordered correctly, then enclosed objects can be checked to make sure the outward direction remains consistent over a surface.

To find out if the vertices in any given polygon are ordered correctly, a point, point 'A', is projected from the center of a polygon in the normal direction of that polygon to a sphere that encloses all of the polygonal objects, to give point 'B' as shown in Figure 3.

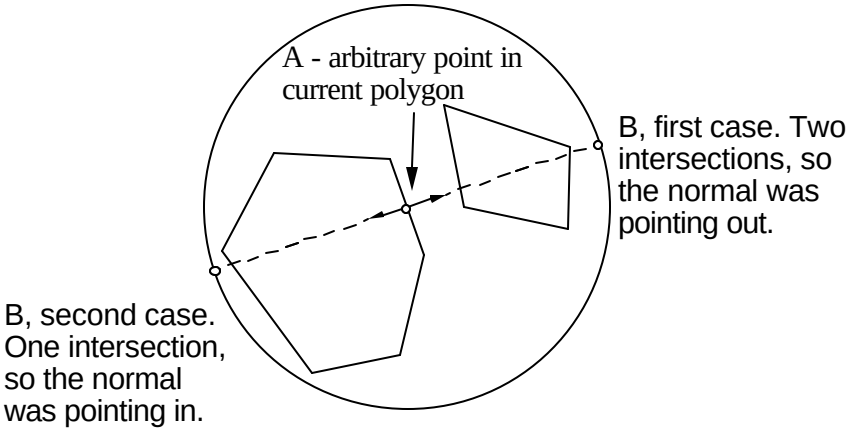


Figure 3: Finding if a polygon's vertices are correctly ordered.
The two cases of point B represent two different ordering conventions.

Then the number of polygons that are intersected by the segment from point A to point B are counted. When objects are enclosed, an even number of intersections implies that the original normal was pointing outward and an odd number of intersections means that the normal was pointing inward.

Determining the Active Polygon and Penetration Depth

Once a collision is detected, one would then like to be able to slide the cursor from one polygon to another to touch the entire object. Several problems arise while trying to do this. First, the active or

current polygon must always be known so that the direction of the force can be determined. Second, while sliding across polygons, if there is a sudden change in magnitude or direction of the normal force, then corners feel sharp and distinct even when there is only a slight angle between the adjoining polygons.

Originally, the transition from one polygon to another was accomplished by finding the distance from the cursor to the three edges of the polygon's normal projection. If the cursor crossed the projection then there would be a new active polygon. The problem with this approach was that the distance from the cursor to the current polygon's surface would change when changing polygons, and a small jerk would be felt even when the normal direction was interpolated correctly as shown in Figure 4a. This is a problem not found in graphics interpolation because a second variable, depth, is included in the overall interpolation. Also, the distance to the edge of the plane would change, which is used in interpolating the direction of the force. And finally, there can be places within an object that are not in any polygon's projection.

Therefore, a different method was determined in which the distances to 'edge' planes rather than the normal planes were found (Figure 4b). An edge plane, for a given edge, is determined from the two polygon vertices defining the edge and a vector that is the average of the normals of the polygons sharing the edge.

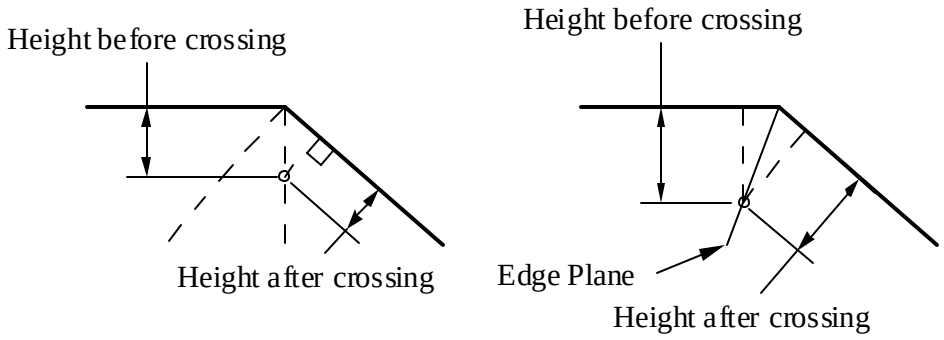
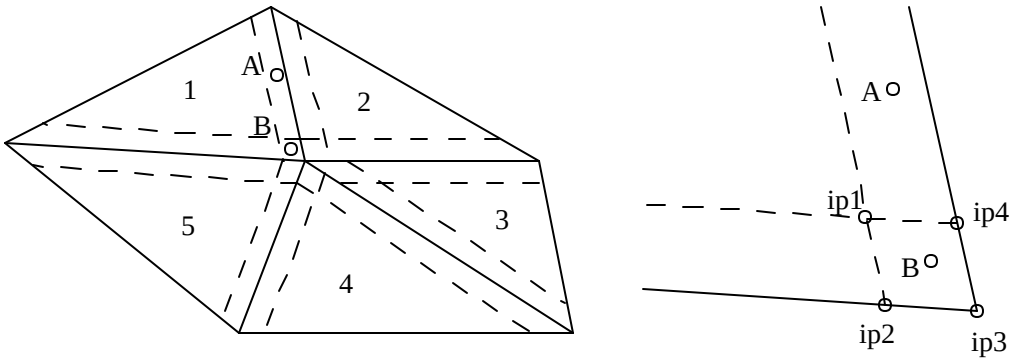


Figure 4: Determination of a change in the current polygon.

The method shown in Figure 4b makes the penetration depth and distance to the edge planes consistent while sliding to another polygon. When a change in the active polygon is detected, the new active polygon can be found by finding the other polygon that contains the two vertices in the edge plane that was crossed. This information can be preprocessed and stored in an array rather than finding it as the cycle runs, which saves on cycle time.

Force Direction Interpolation

In addition to consistent depth of penetration distances, the directions of the forces need to be consistent to keep the edges smooth. This is accomplished by interpolating between adjoining polygons in a way similar to Phong shading in graphics. When the projection of the cursor into the active polygon comes within a fixed distance from the edge planes, the normal direction is interpolated and then normalized.



(a) **(b)**

Figure 5: Interpolation of the normal direction.

The direction at point ‘A’ is interpolated continuously (not necessarily linearly) between polygons 1 and 2. At point ‘B’, the normal is interpolated from 4 points, ip1 through ip4, all of which are normalized. Ip1 is the normal direction of polygon 1. Ip2 is the average of the normals of polygons 1 and 5. Ip4 is the average of the normals of polygons 1 and 2. Finally, Ip3 is the average of the normals of all five polygons. Care must be taken to make sure the direction is continuous while interpolating over boundaries.

Overall, the interpolation over the edges of the polygons presents an interesting psychophysical effect. If the forces are interpolated correctly, then the shape of the object is perceived more from the direction of the forces than from the cursor’s actual position. For example, the facets of a polygonal sphere are easily distinguishable with no interpolation. With the introduction of interpolation, the facets become less noticeable and the sphere feels rounder. As the area over which each facet is interpolated increases, until the forces over the entire facet are interpolated, the sphere appears to ‘fill out’. Although a user is still feeling a faceted sphere, it appears completely round and smooth. This can be a powerful effect as the interpolation can be used to modify the way that an object’s shape is perceived. Additionally, if an edge is supposed to be distinct, then the interpolation can be turned off which will produce a sharp edge.

Acute Edges

If the tool is touching the outside of an acute edge the force will not be interpolated across that edge, which will make the edge feel distinct and sharp, rather than curved. The force is interpolated across the edge in Figure 6a. The force in Figure 6b is taken from the current polygon only. Additionally, there is an issue with acute edges known as the thin-wall problem. This is a common problem in haptic rendering algorithms where the cursor can unintentionally push though a thin wall, such as a knife blade. To handle this problem in the ActivePolygon algorithm, the edge plane that is normally used to transition to a new active polygon, as shown in figure 6a, is not computed or used for acute angles, as shown in Figure 6b.

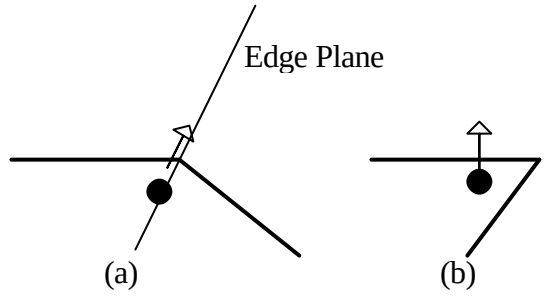


Figure 6: Interpolation force for an exterior acute edge.

If the tool is on the inside of a very acute edge it may travel some distance from the polygon before approaching the edge plane, which can create inconsistencies in the forces. Figure 7 shows a cursor point that has touched an interior edge of an acute angle, and has penetrated into the object.

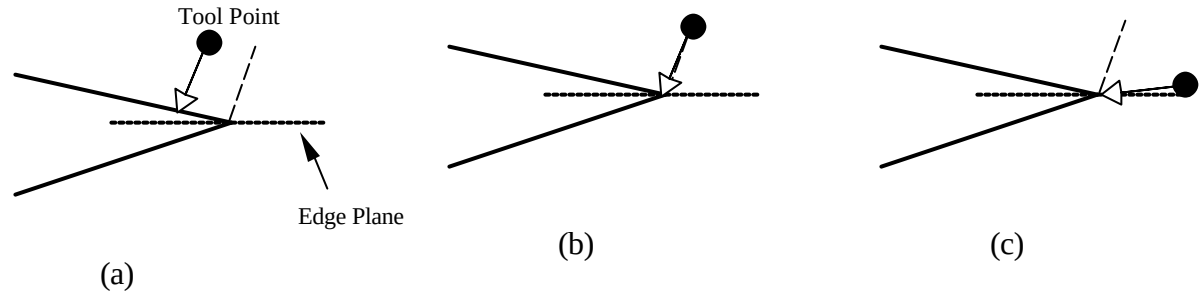


Figure 7: Interpolation force for an interior acute edge.

The point is within an orthogonal projection of the polygon in Figure 7a, so the force is in the polygon's normal direction. In Figure 7c, the tool has moved outside the polygon's orthogonal projection, so the force is directly towards the edge. At the point where the direction of the force changes from the normal direction of the polygon to the edge, as shown in figure 7b, both of those computations are equal, thus giving a consistent force. To increase the speed of the algorithm, acute edges are marked during pre-processing.

Three or More Polygons Sharing a Common Edge

As has been described, the ActivePolygon algorithm utilizes a local region when computing forces. The currently active polygon and the polygons sharing its vertices are used to create forces based on a single point. In order to speed up the algorithm, and reduce processing time, polygonal neighbors are preprocessed. However, when a single edge is shared by three or more polygons, special processing must be done. In this case, the neighbor information is dependent on the side of the polygon. When three or more polygons share a common edge, one side of a polygon has one neighbor, and the other side of the polygon has another neighbor. If an edge has only two polygons attached to it then the front and rear face neighbors are the same.

The side of the polygon which is currently active will determine which of the two neighbors is used for processing.

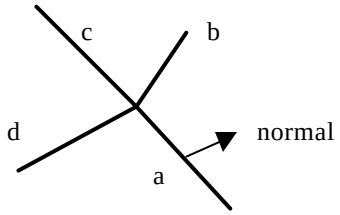


Figure 8: An edge with three neighbors.

Shown in Figure 8 is an edge that is shared by 4 polygons. Polygon a will have polygon b as a front neighbor and polygon d set as a back neighbor. Polygon c will not be a neighbor of a.

Bendable Polygon Algorithm

As the forces are presented to the user, the visual aspects of the virtual object must be consistent with the object. A compliant object should deform as the cursor moves into it. If the object does not deform, then the cursor can be lost visually within it. One way to solve this problem is to simply project the visual cursor, in the direction of the force, to the surface of the object.

However, if the object is very compliant, then this can create a discrepancy between the visual and haptic senses. To solve this, the Bendable Polygon technique is used. When the cursor first touches a polygon, it is split into 6 different polygons as shown in Figure 9. The cursor is projected normally to the plane of the active polygon, and then that point is projected to the edges of the polygon, making base points that are the framework for the approach. The ‘edge base points' move as the cursor moves, always normally projected to the sides of the current polygon. When the polygons are Gouraud or Phong shaded, the edges look smooth and the polygon seems to bend. The effects of the Bendable Polygon technique decrease with increased graphical detail, but for relatively large polygons, the effect works well and allows for lower levels of detail on a deformable object.

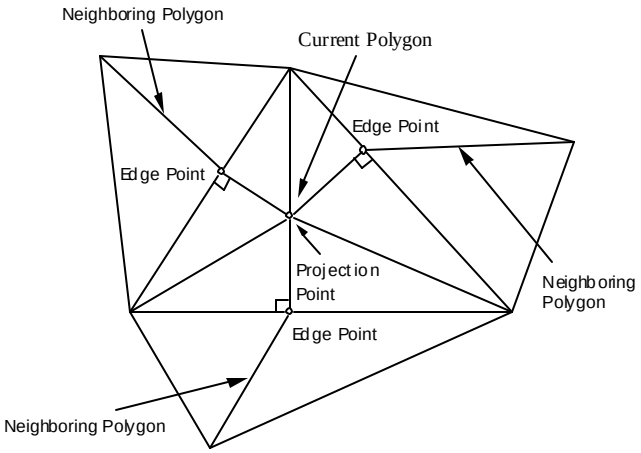


Figure 9: Base Points in the Bendable Polygon algorithm.

The cursor is then connected by a spring and a damper to each of the vertices in the active polygon and to each of the edge points. The cursor does not have any forces applied to it by these springs. All six of those points, in turn are connected to the base points shown in Figure 9, also by springs and dampers.

When the cursor moves into a polygon, the vertices (open circles) are pulled away from their respective base points (filled circles), making the polygon bend as shown in Figure 10.

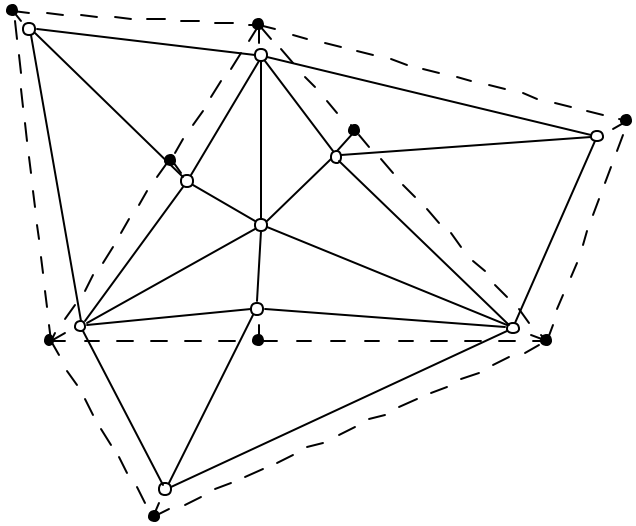


Figure 10: Base Points and Vertex Points in the Bendable Polygon algorithm.

The dashed lines represent the object while it is not deformed, and the solid lines represent the polygons that the user sees after it is deformed. As the cursor moves towards an edge, the springs from the edge points to their respective base points lose strength, so the indentation in a polygon remains consistent even as the cursor moves across different polygons. Different spring constants and different levels of strength reduction give different amounts of indentation (i.e. a small indentation on a water balloon as opposed to a larger indentation on a trampoline). The springs from the corner vertices to their base points work similarly, in that they lose strength as the cursor approaches them, so the indentation remains consistent at the corners of the polygons as well.

Then, each of the vertices throughout the object is connected by springs and dampers to each of the vertices touching it, to give an overall ability of large-scale deformation. In large data sets, the number of calculations would become extremely large, so springs might, for example, only be connected to vertices in the general surrounding area of the cursor, depending on the application, or a finite element analysis can determine the object's deformations.

Because the edge points split the current polygon, each of the 3 neighboring polygons must be split into two polygons as well so that there is no gap (Figure 10). The graphics loop therefore draws 6 polygons in place of the original, 2 polygons in place of each neighboring polygon, and then draws all of the rest of the polygons.

The graphical material effects are accomplished by finding the normals for each vertex, an average of all the normals of the polygons containing that vertex. Then the surface is either Phong or Gouraud shaded according to those normals.

Deformations

The base points in the polygonal algorithm can be given dynamics properties to allow the deformation of an object. This can be accomplished in several ways. Vertices can be given dynamics properties as described above in the dynamics section, with a very small time variable. In this way, the vertices move slowly and create a feel similar to clay. Additionally, the vertices can be moved with a simpler

method of simply displacing them proportional to the force presented. This however does not allow as much flexibility in modifying the feel of the object.

There are several issues that arise when the vertices are allowed to move. First, there must be some constraints applied to the vertex movements so that a polygon does not collapse on itself. This can be done by maintaining a minimum distance for a side on a polygon, or by allow the vertices to move in only a specified direction along a line.

An additional problem comes from the oct-tree based culling which allows real time collision detection. If vertices are allowed to move then the pre-processed tree structure, that describes which polygon should be checked for collisions given the cursor is in a specific leaf node, can be made invalid.

Results

The ActivePolygon algorithm was tested on a Dual PIII 800MHZ computer with 512MB of memory, a Wildcat 4210 graphics card, and a Phantom desktop haptic device. The haptics load was obtained using the GHOST 3.1 Haptic Load program. Object 1 has a complex geometry with many areas that are often troublesome. Objects 2 and 3 were the same object and had the same topology, a relatively simple topology, but differed only in their resolution and polygon counts. Object 4 had a large polygon count. There were several aspects of the results that were significant. First, the haptic load averages and peaks were consistent across objects with varying polygon counts. This was expected as the computations are based on a pre-processed data set and are computed locally so the overall size does not affect the local computations. Second, the haptics were stable across varying and complex geometries. Third, the load times increased as the objects had more polygons. This also was expected as there was therefore more data to be preprocessed. A majority of the load time is due to the preprocessing. After the preprocessing is done once, the preprocessed data can be saved with the object, and the load time can be perceptually eliminated. We additionally tested an object with over 1 million polygons. The haptic load peak occurred when initially touching or leaving that object, but maintained a consistent average.

Table 1: ActivePolygon results

Object	Polygons	Load Time (sec.)	Visual FPS	Haptic Load Avg.	Haptic Load Peak
1	5706	1.03	21.33	20%	20%
2	134232	19.54	6.76	15%	20%
3	239694	33.52	3.8	15%	20%
4	1063452	195	0.98	15%	80-100%

Table 2: Comparison with GHOST polygonal renderer

Object	Polygons	Load Time (sec.)	Visual FPS	Haptic Load Avg.	Haptic Load Peak
1	5706	0.9	15	20%	30%
2	134232	11.74	6	85%	100%
3	239694	17.12	6	70%	70%
4	1063452	220		Unstable haptics	

We also compared the ActivePolygon algorithm to the TouchVRML polygonal renderer contained in the GHOST 3.1 API from SensAble Technologies. The listed TouchVRML Visual Frames Per Second are highly qualitative and were obtained by viewing and estimating. In objects 1, 3, and 4 using TouchVRML, the phantom would have force kicks in certain manipulation situations. Objects 3 and 4 had unstable haptics in the TouchVRML application.

Overall, the ActivePolygon algorithm worked very well. Further research is being done to continue to extend its functionality and scope, add modifications such as haptic textures, and integrate it into software applications. The source code for the ActivePolygon algorithm is available in the e-Touch library on the e-Touch web site, www.etouch3d.org.

Acknowledgements

The writers would like to acknowledge Walt Aviles, Sandia National Laboratories, Arthurine Breckenridge, George Davidson, Tom Caudell, and Tom Furness for their help in the development of this work.

References

[1] K. Salisbury, D. Brock, T. Massie, N. Swarup, and C. Zilles, “Haptic Rendering: Programming Touch Interaction with Virtual Objects”, ACM Symposium on Interactive 3D Graphics, Monterey, CA, 1995.

[2] T. Massie and K. Salisbury, “The Phantom Haptic Interface: A Device for Probing Virtual Objects”, Proceedingsmof the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, Chicago, IL, 1994.

[3] SensAble Technologies Inc., “The PHANToM” literature from SensAble Technologies.

[4] C. Zilles, J. Salisbury, “A Constraint- based God-object Method for Haptic Display”, IEEE/RSJ International Conference on Intelligent Robots and Systems, Human Robot Interaction and Cooperative Robots, 1995.

[5] P. Buttolo, B. Hannaford, B. McNeely, “Introduction to Haptic Simulation”, Tutorial 2A, IEEE/VRAIS tutorial notes, March, 1996.

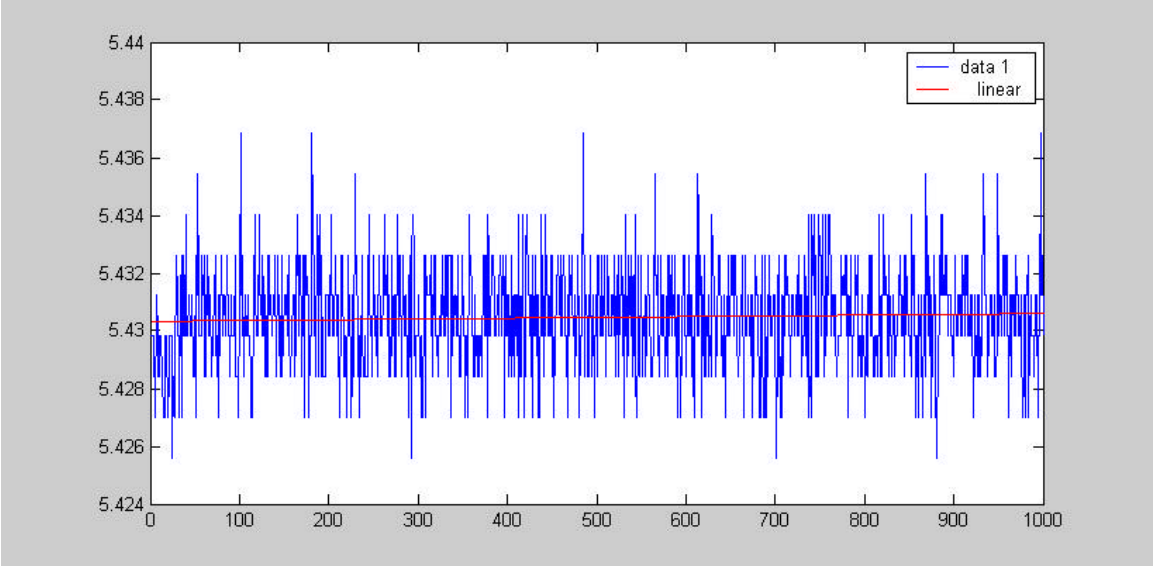
Dealing with Desktop Gimbal Noise

Karl Reinig Chris Lee

Introduction

The gimbals of the desktop PHANToM introduce considerable noise into the measured state of the stylus. When the virtual tool tip is represented by a point located at the origin of the gimbal, the noise goes unnoticed. However, when the virtual tool tip is projected away from the origin of the gimbal or the tool possesses non-trivial extent such as a ray or cylinder, the noise noticeably degrades both the haptic and graphic display of the tool. If the tool is being used as a virtual camera, the noise will introduce jitter into the graphic display of the virtual environment. This paper discusses some of the problems and work-arounds, when attempting to filter the noise.

The following figure shows one of the gimbal angles, as reported by the GHOST method getGimbalAngles.



Throughout the one thousand samples, the stylus was resting on a solid surface. The resolution of the encoder, approximately .0015 radians or .086 degrees, can be seen in the discrete steps of the samples. The graph also shows the noise to be about eight times the resolution.

Consider the consequences of projecting the virtual tool ten centimeters from the gimbal origin. Ten centimeters times the tangent of .0015 (the gimbal resolution) results in a positional resolution of .015 cm. But the noise results in a positional jitter of approximately 1.2 mm. This is unacceptable for most applications.

Filtering the Noise

Creating an optimal filter is not the point of this paper. Instead, we present some of the problems we encountered while implementing a simple filter and discuss our current work-around. Consider implementing a filter that simply averages the signal by adding equally weighted samples and dividing by the number of filter samples. The resulting low-pass filter will scale the excursions due to the noise by the inverse of the number of filter samples. It will, of course, also degrade the high frequency fidelity of the virtual tool.

Let $?_i$ be the sample taken i steps before (for example, the current sample would be $?_0$).
Let $F(?)$ be the result of filtering the $?_i$.
For our simple filter,

$$F(?) ? \frac{1}{n} \sum_{0}^{n?1} ?_i$$

In GHOST, the gimbal state is available as the upper 3 x 3 elements of a 4 x 4 transform. The most straightforward implementation of the filter would perform the average on each of the elements of the 3 x 3 array. However, this is more than just wasteful since the resulting transform would no longer be a rotation matrix. At the very least, you would have to renormalize its elements. This is similar to the problem of using matrices to represent camera motion.

Ideally you would filter the gimbal noise directly and then use it to form the PHANToM transform. Unfortunately, while GHOST does give direct access to the gimbal angles, it does not give access to the methods that create the PHANToM transform. Since the rotational components of the PHANToM transform contain positional information, this is not a trivial transform.

It would be helpful if the GHOST team made a method available to combine gimbal angles and position information to create the transform. This is not necessarily straightforward since some of the required information probably only exists at the driver level. What follows is a less than ideal solution for the interim.

GHOST does make a set of Euler angles available from the PHANToM transform. The angles are the rotations about the x, y, and z axes that would put the stylus in its current orientation. The Euler angles can be used to recreate the PHANToM transform and are therefore candidates for filtering. Unfortunately, the transform that gets the Euler angles from the PHANToM transform does not produce a continuous set of Euler angles. There are places in which each of the Euler angles jump by 2PI. Since a jump of 2PI about any axis has no effect on the orientation, the ambiguity generally goes unnoticed. However, the result of filtering an angle that has 2PI jumps is an angle that makes a quick trip around the quadrants. This causes completely unacceptable rapid movements in the virtual tool. The method described here recognizes the occurrence of a 2PI jump and removes it from the current sample. Note that the gimbal angles themselves introduce no such ambiguity.

Implementation

A brute force implementation of the averaging filter could be accomplished by keeping an array of samples. Samples would be added to the array using a cyclic index that runs from 0 to n-1. At each time step, the filtered value could be found by summing the elements of the array and dividing by its length. If the averaging is to be performed over 100 steps, each time step would contain the summing of 100 elements.

The summation can be eliminated by developing a recursive implementation. It can be shown that the difference between the filtered samples at consecutive times steps is just one over the length of the filter times the difference between the current sample and the oldest sample. Therefore instead of summing the elements, we need only find the difference between the current sample and the oldest sample, scale it by the inverse of the number of steps, and add it to the previous filtered value.

To eliminate the problems caused by the 2PI jumps, compare the current sample with the previous sample. If the current sample is more than PI larger than the previous sample, simply subtract 2PI from the current sample. If the current sample is more than -PI smaller than the previous sample, simply add 2PI to the current sample. As long as the magnitude of the actual change in the angle is less than PI, there should be no ambiguity.

The PHANToM transform can be constructed from the Euler angles using the following matrix, where X, Y, and Z are the first, second, and third elements of the Euler vector.

$$\begin{bmatrix} \cos Y * \cos Z & \cos Y * \cos Z & -\sin Y & 0.0 \\ \sin X * \sin Y * \cos Z - \cos X * \sin Z & \sin X * \sin Y * \sin Z + \cos X * \cos Z & \sin X * \cos Y & 0.0 \\ \cos X * \cos Z * \sin Y + \sin X * \sin Z & \cos X * \sin Y * \sin Z - \sin X * \cos Z & \cos X * \cos Y & 0.0 \\ posX & posY & posZ & 1.0 \end{bmatrix}$$

Summary

The gimbals of the PHANToM desktop introduce unacceptable noise into the state of the stylus. The information required to transform the gimbal angles into a useful PHANToM transform are not made available. A solution is to decompose the PHANToM transform into Euler angles, filter them (watching for 2PI jumps), and then reconstruct the PHANToM transform.

Expanding the Haptic Experience by Using the PHANToM Device as a Camera Metaphor

Joan De Boeck, Chris Raymaekers, Karin Coninx
Expertise Center for Digital Media
Limburg University Center
Wetenschapspark 2, B-3590 Diepenbeek, Belgium
{joan.deboeck, chris.raymaekers, karin.coninx}@luc.ac.be

Abstract: *Many applications that support force feedback make use of a haptic device (such as the PHANToM) used for pointing operations, combined with a second device, mainly used for navigation (such as a 3D mouse). In this research we formally assess the user's performance in a setup in which we use the PHANToM device for camera manipulations. This allows us to eliminate the second device and to free the user of the mental load to drive two different devices. Additionally, when using Sensable's PHANToM as a camera device, we will look into the effect of the additional force feedback.*

1. Introduction

Most desktop haptic applications consist of a two-device setup. The haptic device, mostly manipulated with the dominant hand, is used for pointing and manipulation operations. A second device (typically a 3D mouse) is used for navigation operations.
In the context of our research, very little can be found in literature about integrating forces in camera manipulations. In the early 90's efforts have been made in defining the best navigation metaphor [1] for a certain task in a virtual world. *Flying vehicle* and *Scene in hand* are the most commonly known. Other work has been done in improving navigation and wayfinding methods in virtual environments [2]. In some research systems hand-held miniatures [3] or Speed-coupled Flying [4] are presented to facilitate the user's interaction. A usability test by T.G. Anderson [5] provides evidence that additional force-feedback results in better performances when compared to the 2D navigation interface of CosmoPlayer. In this paper, based on Anderson's work, we introduce a fairly new variant on the "Eyeball in hand" navigation metaphor presented in [1] and focus on the comparison with the LogiCad Space Mouse [6].
In the next section we explain our "Camera in Hand" metaphor as a variant of the "Eyeball in hand" navigation metaphor. Afterwards, the experimental setup used to assess users' performance when navigating with this metaphor is described. The results of the formal evaluation are summarized and discussed. Finally, conclusions with regard to the usefulness of the proposed metaphor are formulated.

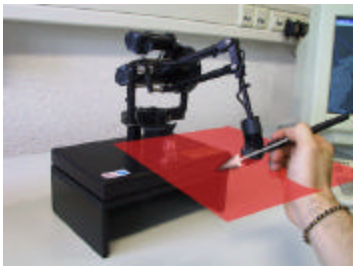


Fig 1. PHANToM as a camera device with virtual guiding plane

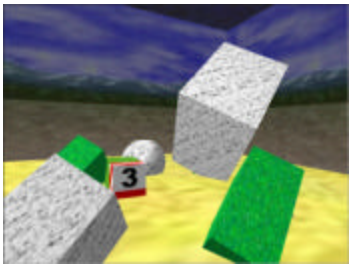


Fig 2. Virtual arena in which users have to locate the number

2. Eyeball in Hand/Camera in Hand

A possible implementation of the *eyeball in hand* metaphor is to use a Polhemus tracker as a virtual eyeball, which can be moved about the virtual scene. However, this manipulation method appeared to imply a confusing mental model in which disorientation is a common problem. In former research in our lab the *Eyeball In Hand*-metaphor has been extended to a MicroScribe device [7][8]: by moving the MicroScribe's stylus with the non-dominant hand, the virtual camera is repositioned. By defining the viewpoint in such a manner that it matches the direction of the stylus, disorientation will be avoided. As we are not actually handling an eyeball anymore, but rather a pen-like object, we will now call this extension the *Camera In Hand* Metaphor.
This paper adopts the latter metaphor for the PHANToM haptic device, and extends it by applying additional force-feedback. Informal testing taught us to set up a horizontal virtual plane (as shown in fig. 1) as the most useful feedback. This allows the user to easily walk forth and back in this plane. When changing the viewpoint's altitude the user has to act against the resistance of the PHANToM. A formal experiment, described below, was set up in order to formalize and detail the results obtained by informal testing.

3. Experimental Setup

The aim of our research was to formally compare this new metaphor and camera device to another existing 3D device. For this comparison condition we have chosen to use the LogiCad SpaceMouse in a "Flying Vehicle" metaphor.
Twenty-two volunteers with mixed experiences in virtual environments participated in the experiment. Most of the subjects are in their late twenties or early thirties, although 4 of them were above the age of 40. All subjects were right-handed and one third of the population was female.
All of the participants had to navigate in a virtual arena to locate and read a digit on a red-white coloured object (see fig. 2). This test had to be performed in three conditions; each condition consisted of 15 trials. The first condition measured performance with the SpaceMouse, the second looked at the PHANToM device without force feedback and finally the PHANToM device with force feedback has been tested. To eliminate transfer effects, the order in which to take the experiments was counterbalanced. During each trial the elapsed time and the total traveling distance had been logged. Finally, at the end of the test a comparative questionnaire had to be filled-up by the subjects. We have to note that one of our 22 test persons had trouble in performing the tasks in all conditions. Since his results exceeded 12 times the standard deviation, we have omitted those values.

4. Results

Chart 1 shows us the median values of the completion times of all subjects, per trial in each condition, which gives us a first impression of the results.

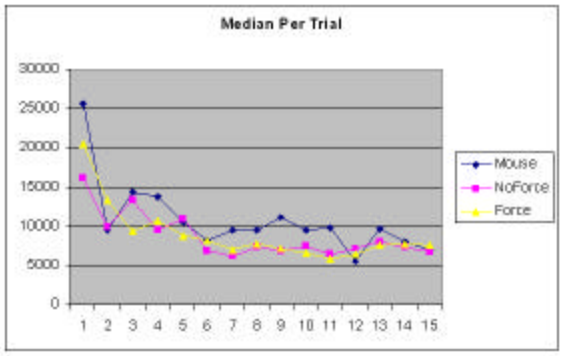


Chart 1. Completion Time (ms). Median values per trial

In our further analysis[1], we consider the first 5 trials for adaptation to the proposed input devices, and so leave them out of our computations[2]. As can be seen from chart 1 and table 1, the average completion time in both PHANToM conditions is slightly better than the SpaceMouse. With a P-value of 0.12, there is no significant difference, however.

[1] Using ANOVA

Mouse	13333 ms
PHANToM Force	10256 ms
PHANToM NoForce	9499 ms

P-Values	
Condition[Mouse-PH no]	0.1231
Condition[PH Fo - PH no]	0.8241

Table 1. Averages and P-values over all subjects

Because of the relative heterogeneity of our population, we have divided all subjects in four categories depending on their experience in 3D navigation: no, little, much and very much experience. Statistically, the groups with little, much and very much experience behave the same. Therefore, in our further analysis, we consider two levels of experience: novice (users without any 3D navigation experience) and experienced (all the others).
If we look at the average completion times in table 2, we can see there's still no significant difference between any of the conditions in the experienced group. However, we now notice a strong significant difference in completion times between the SpaceMouse and the PHANToM conditions among the novice users.

	Novice Users	**Experienced Users**
Mouse	17263 ms	9760 ms
PHANToM Force	9381 ms	11060 ms
PHANToM NoForce	9007 ms	9941 ms

P-Value		
Condition[Mouse-PH no]	<.0001	0.4922
Condition[PH Fo - PH no]	0.0507	0.2636

Table 2*. Averages and P-values per category*

A subjective questionnaire, filled up by all subjects after the test, shows us that experienced users significantly prefer the SpaceMouse over the PHANToM. On the other hand novice users choose one of both PHANToM conditions.

	Novice	Expert
Mouse	2	10
NoForce	5	1
Force	3	0

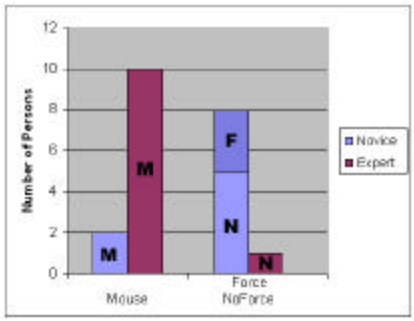


Fig. 3. *Subjective preference per category*

5. Discussion

As can be seen from table 2 experienced users objectively do not perform different in one or the other condition. If we look at the measurements of the novice users, we see a dramatic improvement when using the PHANToM. Compared to the values of the experienced category, we can notice that the values of the novice users using the PHANToM condition are similar. This means we can conclude that our *Camera In Hand* metaphor provides a possibility for the inexperienced user to perform equally to their experienced colleagues.
However we can also conclude that the addition of force-feedback, which implements a horizontal guiding plane, doesn't offer any advantages. There's even advantage for the no-force condition, though the difference is insignificant.

---

[2] This is supported by our results, as can be seen in chart 1.

As the performances of the experienced users are similar in all conditions, we have to ask why they collectively choose for the SpaceMouse condition. Our experienced users all spend several hours a day on a computer and all have their 3D experience playing games with mouse and keyboard. For that reason we suspect those users to have certain expectations and so feel more familiar with the SpaceMouse. In addition, some of those users report the limited workspace and tiring pose when using the PHANToM as a disadvantage.

6. Conclusion and future work

In this work we presented a 3D camera metaphor using the PHANToM device with and without force feedback. This metaphor can eliminate the use of a second input device in a haptic setup. The performances of those conditions have been measured and compared to the navigation with a SpaceMouse in a formal usability test. As a conclusion we can state that, using the PHANToM, novice users act in the same way with the *camera in hand* metaphor as the experienced users. When those users are using the SpaceMouse there is a strong performance penalty. However, experienced users mostly choose for the SpaceMouse, while they perform equally in all conditions. Finally, we also can conclude that additional force feedback in a sense of an additional guiding plane doesn't offer any benefits in this test.

We believe the *camera in hand* metaphor will be of interest to introduce the novice user into 3D environments, but it can also be useful when manipulating a scene that is rather centralized in a limited volume. Although additional force feedback doesn't seem to offer any benefits, we think, dependent on the task, additional stability can be obtained by finding a appropriate force factor, which is possibly somewhat smaller than the resistance force in our test.

In our future work, we want to evaluate the effect of additional functionality to step out of the limiting workspace of the PHANToM. This can be achieved by e.g. homing the PHANToM without changing the virtual camera, or by moving the virtual camera when the users pushes the outer limits of the PHATNoM's workspace [5].

7. Acknowledgements

Part of the work presented in this paper has been subsidized by the Flemish Government and EFRO (European Fund for Regional Development).

We also want to thank the people of CIT Engineering and Porta Capena and all other volunteers for their participation in this test.

8. References

[1] C. Ware, S. Osborne. Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. Computer Graphics 1990 Vol 24 Nr 2

[2] G.A. Satalich. Navigation and Wayfinding in Virutal Reality: Finding Proper Tools and Cues to Enhance Navigation Awereness. http://www.hitl.washington.edu/publications/satalich/home.html

[3] R. Pausch, T. Burnette. Navigation and Locomotion in Virtual Worlds via Flight into Hand-Held Miniatures. Computer Graphics 1995, Annual Conference Series, p 399-400

[4] D.S. Tan, G.G.Robinson, M. Czerwinski. Exploring 3D Navigation: Combining Speed-coupled Flying with Orbiting. CHI 2001 Conference Proceedings p. 418-425, Seattle, Washington, USA

[5] T.G. Anderson. FLIGHT: An Advanced Human-Computer Interface and Application Development Environment. Thesis Master Of Science, 1998, University of Washington.

[6] LogiCAD SpaceMouse; http://www.logicad3d.com/products/Classic.htm

[7] T. De Weyer, K. Coninx, F.. Van Reeth Intuitive Modelling and Integration of Imaginative 3D Scenes in the Theatre, Proceedings VRIC 2001 p 167-173

[8] Immersion Microscribe 3D: http://www.immersion.com/products/3d/capture/overview.shtml

Fast Haptic Rendering of Complex Objects Using Subdivision Surfaces

Chris Raymaekers Koen Beets Frank Van Reeth

Expertise Centre for Digital Media, Limburg University Centre
Wetenschapspark 2, B-3590 Diepenbeek, Belgium
{*chris.raymaekers, koen.beets, frank.vanreeth*}*@luc.ac.be*

Abstract

Haptic rendering of meshes of arbitrary topology is a difficult and time-consuming process. This paper presents the use of subdivision surfaces in order to solve this problem. An overview of subdivision surfaces is given and the algorithms needed to calculate the surface contact point are discussed. We will show that this algorithm is faster than traditional algorithms.

Introduction and Related Work

Haptic rendering of complex objects needs a lot of calculating power. Even with modern computers, limiting the number of calculations that need to be made on a complex object can be very difficult. Two approaches are frequently used: mathematical representations of the objects, and polygonal models. The GHOST SDK (SensAble, 2001) uses both methods: a mathematical representation is used for simple shapes, such as cubes and cones, while arbitrary meshes are represented by a polygonal model.

Most of the time, only a limited number of polygons is used in order to keep the calculations within the 1ms interval of the haptics loop. However, this also limits the precision of the model. An alternative is the use of mathematical representations, such as NURBS. This however introduces other problems: representing models of arbitrary topology is for instance very difficult.

In our research, we would also like to deform the objects. Using NURBS, the seams of the patchwork can become visible during deformation. This problem also occurs in computer animation and has been solved by using subdivision surfaces. A famous example is the Pixar movie "Geri's Game" (DeRose et al., 1999).

Subdivision Surfaces

A subdivision surface is a surface that is defined as the limit of a series of refinements $M_1, M_2, \ldots$, starting from an original control mesh M_0. Because of this property, they support level-of-detail. Since they can also be used to efficiently represent objects of arbitrary topology, and they can be modified easily to support features such as creases and boundaries, it is no surprise subdivision surfaces are already used in computer graphics and computer animation. Figure 1 shows a control mesh and the first refinement.

A wide variety of subdivision schemes for surfaces exist, with an equally large variety in properties. Two of the most well-known subdivision schemes for surfaces are the Catmull-Clark scheme, which works on quadrilateral meshes, as described in (Catmull and Clark, 1978), and the Loop scheme, which is triangular (Loop, 1987). The types of surfaces generated by these schemes differ, but the general principles of subdivision surfaces remain the same. In our research, we use the triangular loop scheme because triangular polygons are very well suited for modelling freeform surfaces, and they can be easily connected to an arbitrary configuration . For a detailed explanation of subdivision and subdivision surfaces, we refer the interested reader to (Zorin and Schröder, 2000).

The Loop scheme, based on the three-dimensional box spline, is an approximating face-split scheme for triangular meshes, invented by Charles Loop. The resulting surfaces are C^2 everywhere except at extraordinary vertices — with a valence different from 6 — where they are C^1.

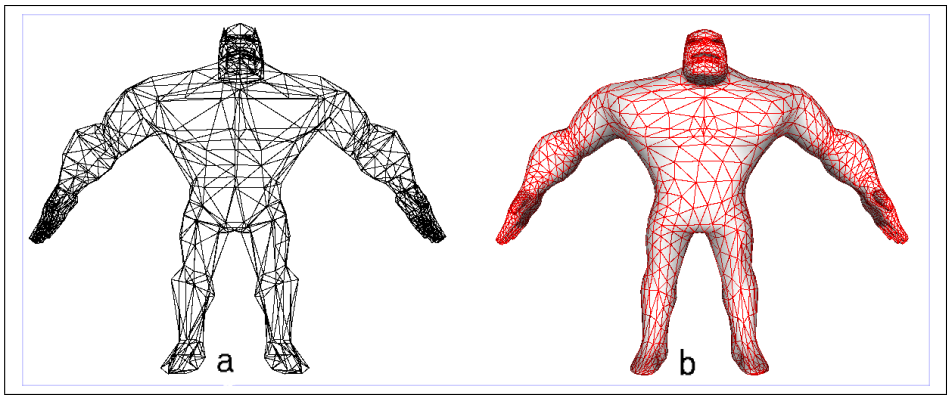


Figure 1: Loop control mesh and first refinement

An iteration of the scheme consists of two stages. In the first stage, known as the splitting stage, a new vertex is added in the middle of each edge, and both the old and new vertices are connected to form 4 new triangles for each old triangle. In the smoothing stage, all vertices are averaged with their surrounding vertices. This smoothing step, together with the weights used, is visualized in figure 2. Loop's original choice for β was $\beta = \frac{1}{k}(\frac{5}{8} - (\frac{3}{8} + \frac{1}{4}cos(\frac{2\pi}{k})^2)$, where k is the valence of the central vertex, but other choices are possible as well (Warren, 1995). Figure 2 also shows that the support — which is the region over which a point influences the shape of the limit surface — of the Loop scheme is small and limited.

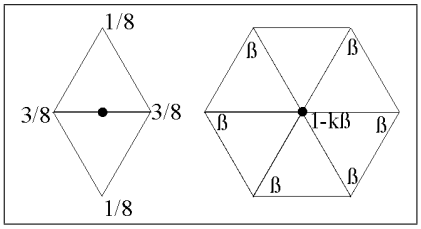


Figure 2: Subdivision Mask of the Loop Scheme

Calculations using Loop Surfaces

During haptic rendering of polygon meshes, all of the object's faces generally need to be processed. This is no longer necessary with subdivision surfaces. The multiresolution properties of subdivision surfaces can be exploited so that only the control mesh has to be processed as a whole. In the processing stages of the following, more detailed, subdivision levels, the results of the previous test are used, thus leading to a smaller number of polygons that have to be processed. This leads to a huge increase in speed.

As mentioned in the previous section, a subdivision surface is defined as the limit of a series of surfaces. This gives rise to two interesting properties:

- Every face at level $n-1$ can be linked to four faces at subdivision level n.
- As can be seen from figure 2, the new co-ordinates of a vertex are influenced by the surrounding vertices. Using the loop subdivision scheme, most vertices have a valence of 6, so 6 vertices are needed to calculate the new co-ordinates. Generalizing this for a triangle (figure 3), each vertex of the triangle is influenced by its 6 neighboring triangles. Since a number of these neighboring triangles are shared, 12 neighboring triangles are needed to calculate the new co-ordinates of each of the triangle's vertices. The grey triangle in figure 3 is the triangle that is subdivided.

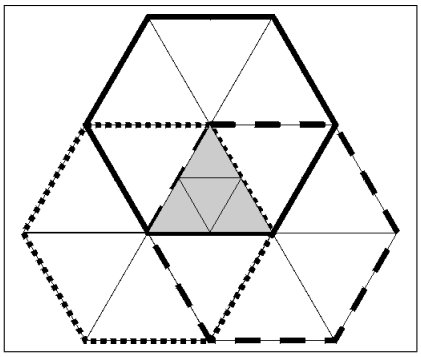


Figure 3: Area which influences a single triangle

The next two sections explain the 2 problems that need to be solved. In both cases the algorithm starts from the control mesh at level 0, leading to an accuracy up to an arbitrary subdivision level n, which contains 4^n times as much triangles as level 0 does.

Performing the inside-outside test

Consider a control mesh M_0, consisting of a triangles. The following steps describe the algorithm that checks wether a point p lies inside or outside the object.

1. Shoot a semi-infinite ray, starting at point p.
2. Select all the intersected faces and the face closest to the point being tested.
3. Extend this selection by including all triangles in the 1-neighborhoods of all vertices of the intersected faces.
4. Replace all the selected faces by their subdivided children. The number of triangles is multiplied by 4. Again test all triangles in this selection for intersection with the ray.
5. Check wether the number of intersections found is different from the previous number. This can happen because the refined meshes "shrink" in areas where the control mesh is convex. In concave regions, the refined meshes grow outside of the control mesh.

   If the number has changed, go to step 7, otherwise proceed with step 6.
6. If the required subdivision level is not yet reached, go to step 2.
7. If the number of intersections is odd, the point lies inside the polymesh. Otherwise it lies outside the polymesh.

Calculating the SCP

Using the results of the previous calculations, the SCP can be calculated.

1. If in the previous algorithm the required subdivion level is found go directly to step 7.
2. Select those faces from the selected faces of the current subdivision level which are closest to the point being tested.
3. Extend the selection to faces which contain vertices in the 1-neighborhood of all vertices in the selected faces.
4. Replace the selection with its next subdivision level.

5. Select the closest face from the previous selection.
6. If the required subdivision level is not yet reached, go to step 3.
7. Calculate the exact intersection point of the closest face. This is the SCP.
8. For each vertex of the triangle the limit normal vector is calculated by taking the vector product of the following two vectors.

$$t_1 = \sum_{i=0}^{k-1} \cos\frac{2\pi i}{k} p_i \qquad t_2 = \sum_{i=0}^{k-1} \sin\frac{2\pi i}{k} p_i$$

The normal in the SCP is calculated by interpolating these three normals.

Comparison

Using subdivision surfaces, only a small number of triangles need to be processed. Consider again the control mesh M_0, consisting of a triangles. At level 0 a intersection tests have to be performed. If k intersections are found (k is typically a very small number), in each step these triangles and their neighboring triangles need to be subdivided. In the worst case scenario, where the neighboring zones are al disjunct, $k*(1+12)*4 = k*52$ intersection tests have to be performed for each subdivision level. The maximum number of tests (if the test is not successful before reaching level n and all zones are always disjunct) is $a + n * k * 52$ (please note that the subdivision process is performed in preprocessing stage) . If a "normal" polymesh with the same number of polygons has to be checked, $a * 4^n$ triangles would have to be checked.

For instance, suppose a control mesh, consisting of 100 triangles, is checked until level 4 (a typical level) and 5 intersections are found. This leads to $100 + 4 * 5 * 52 = 1140$ checks. The equivalent polymesh consists of $a * 4^4 = 25600$ triangles which leads to more than 20 times as much checks.

When using adaptive subdivision, where the fact if a triangle is subdivided depends on the surface area of the triangle (compared to the other triangles of the mesh), even a smaller number of tests is needed.

Conclusions

We proposed a technique for fast haptic rendering by using subdivision surfaces, which can greatly improve speed. Also, by using subdivision surfaces, we can evaluate objects of any topology up to an arbitrary refinement level.

Acknowledgments

Part of the work presented in this paper has been funded by the Flemish Government and EFRO (European Fund for Regional Development).

References

Catmull, E. and Clark, J. (1978). Recursively generated b-spline surfaces on arbitrary topological meshes. *CAD*, 10(6):350–355.

DeRose, T., Kass, M., and Truong, T. (1999). Subdivision surfaces in character animation. In *Proceedings of the SIGGRAPH 1999 annual conference on Computer graphics*, pages 85–94, Los Angeles, CA, USA.

Loop, C. (1987). Smooth subdivision surfaces based on triangles. Department of mathematics, University of Utah, Utah, USA.

SensAble (1996–2001). *GHOST Programmers Guide*.

Warren, J. (1995). Subdivision methods for geometric design. Rice University.

Zorin, D. and Schröder, P. (2000). *Subdivision for Modeling and Animation*. Number 23 in Course Notes for SIGGRAPH 2000. ACM.

Scene Complexity: A measure for real-time stable haptic applications

Eric Acosta, Bharti Temkin
Department of Computer Science, Texas Tech University
Bharti.Temkin@coe.ttu.edu

Abstract

We discuss real-time issues of scene-complexity in order to frame a technical envelope for stable haptic applications. How large a scene can a haptic application support? Specifically, we try to determine the largest stable haptic scene possible when GHOST is used to develop an application. The scene consists of a number of non-overlapping primitives or polymesh objects. The scene complexity is measured as a number of primitive objects or polygons in polymesh objects that allow stable haptic interactions.

The initial data collected indicates that earlier versions of GHOST allow for more complex scenes. Each later version seems to reduce the number of objects that can be included in a scene. For example, Version 1.2 allows for inclusion of 1100 non-overlapping spheres in a stable haptic application. Version 2.0 allows only 770 (almost 30% less) such objects, while version 3.0 allows only 600 (almost 45% less). The system used for data collection is a Pentium III 500 MHz computer with 256 MB of RAM. This machine has an nVidia Riva TNT2 Ultra video card with 32 MB of memory and runs Windows NT Workstation 4.0 with Service Pack 6.0.

In order to quantify and understand these observations, we have developed an application that estimates the distribution of resources used within the haptic loop; the fractions used by collision detection, graphics, and haptic processes. This study provides a method for performance analysis of haptic systems. Scene complexity plays a key role in the creation of haptic applications by providing critical data needed to estimate the feasibility and performance limits for generic haptic environments. It thus should have a broad impact on the design of haptic applications.

1. Introduction

Scene complexity is a measurement on how complex a scene can get and still allow for stable haptic interactions. The real-time performance is typically reduced in proportion to the increase in the complexity of the scene. In fact, it is well known that the time to compute haptic interactions increases with the number of polygons. The approach taken in previous work was to make the haptic servo loop rate essentially independent of the number of polygons [1]. However, this invariance is obtained only after the contact is made with an object and while the object is being touched in the neighborhood of the proxy, while the proxy remains inside the object, making this an efficient haptic rendering technique for a single polymesh object. In this paper, the scene complexity is measured as a number of primitive objects or polygons in polymesh objects. By understanding the limits imposed by the complexity of the scene, we can estimate the feasibility and performance limitations of generic haptic environments.

First, we consider the greatest lower bound (GLB), the highest scene complexity for which the application always runs with stability and no errors. Next we consider the least upper bound (LUB), where some instability and errors are allowed to occur. When the number of objects is between GLB and LUB the application runs some of the times and produces errors or instabilities at other times. When the complexity exceeds LUB of the scene, the application produces errors instantly after haptics is initialized. This establishes a scene complexity stability range for haptic applications, as shown in Figure 1.

Figure 1: Scene complexity stability range.

2. Primitive Objects: Spheres and Box complexity tests

To establish the scene complexity, our application allows a tester to specify the number of primitive objects (in the x, y, and z directions) in the scene. The application automatically creates and spaces the objects to prevent them from overlapping as shown in Figure 2. The use of non-overlapping objects eliminates other factors affecting haptic load (hload) [2]. Hload is the time required to complete a haptic loop. Our application records the hload data with 10^{-3} ms precision and stores it in a file. In order to estimate the time distribution of the graphics, haptic, and collision detection tasks [3] within the 1 ms haptic loop, hload can be measured in several modes: touching (T) or not touching (NT) an object, graphics on (G) or off (NG), and removing geometry from the haptic scene graph (NH). By removing the geometry branch "hapticScene", as seen in Figure 3, the geometry can be eliminated from the haptic scene graph. The NH mode represents the time required to perform other duties of the haptic loop, such as the device position query and the scene graph traversal, without having to perform any collision detection on the geometry objects themselves.

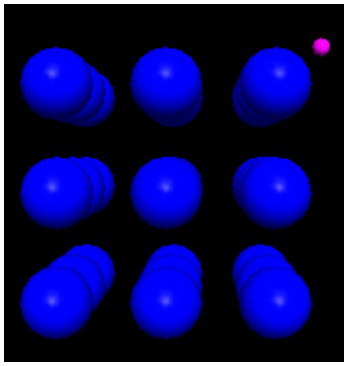


Figure 2: Spheres in a 3x3x3 (x,y,z) fashion.

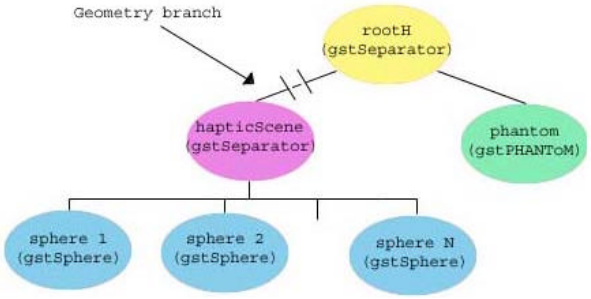


Figure 3: Sample scene graph representation.

The test data was collected five times for a combination of modes between G or NG, T or NT, and NH (G_T, G_NT, NG_T, NG_NT, G_NH, NG_NH). Starting with 8 (2x2x2) objects, the number of objects is increased by one in each direction (e.g. next test has 9(3x3x3) objects), until an error is generated for exceeding the 1 ms time constraint of the haptic duty cycle. At this stage, the number of objects is slowly decreased to identify the GLB. Next, the number of objects is increased to find the LUB, a point at which the application instantly produces an error.

Tests were conducted using three different versions of GHOST (1.2, 2.0, and 3.0) to evaluate performance changes from version to version. The GLBs and LUBs for the different versions are summarized in Table 1. The GLB between version 1.2 and 2.0 is reduced by about 30% and reduced by another 22% between versions 2.0 and 3.0 for spheres. Test results for version 3.0 are displayed in Figure 4. From the tests, we calculated that on average displaying graphics increased the hload by about 2-5% and touching an object increased it by about 7-18%. Our test results indicate that earlier versions of GHOST allowed for more complex scenes. Each subsequent version reduced the number of objects that could be included in the scene for a stable haptic application. The GLB between versions 1.2 and 2.0 is reduced by about 27% and is further reduced by 25% between versions 2.0 and 3.0 for boxes. In general our test show that hload increases

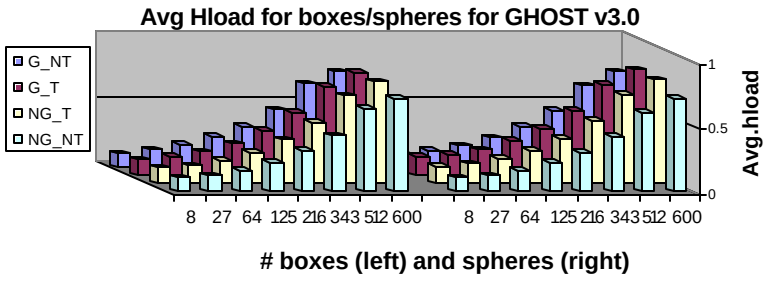


Figure 4: Avg Hload for GHOST v3.0

Version	Object	GLB	LUB
1.2	Box	1100	1188
	Sphere	1100	1200
2.0	Box	800	847
	Sphere	770	847
3.0	Box	600	729
	Sphere	600	729

Table 1: GLB and LUB for GHOST versions

fairly linearly as objects are added to the scene graph for every version of GHOST. Clearly, overhead was introduced from version to version. In case of no graphics and no haptic scene (NGNH), the average hloads were .05ms, .08ms, and .09ms for versions 1.2, 2.0, and 3.0 respectively. The data represents the time required to perform basic duties of the haptic loop, such as device position query and scene graph traversal for nodes other than geometry.

The hload percentage for collision detection is about 82-83% for spheres and 89-90% for boxes, for graphics it is about 3% for both spheres and boxes, and for touching an object it is about 11-13% for spheres and 6-8% for boxes.

3. Polymesh objects complexity tests

For a single polymesh object, a box made up of length, width, and height segments was used. These segments define the resolution of the object and determine the number of polygons that form it. The number of vertices (*nv*) and faces (*nf*) can be calculated as follows, where *l*, *w*, and *h* are the number of length, width, and height segments respectively:

$nv = 2? [(l+1)(h+1) + (w-1)(h+1) + (w-1)(l-1)]$

$nf = 4?[(l?h) + (w?h) + (w?l)]$

Figure 5: Polymesh box

Figure 5 is a wire frame rendering of the box showing how the object is made of triangular polygons. Starting with a 10x10x10 segment (602 vertices, 1200 polygons) box, the numbers of segments (length, width, and height) are increased by 10 for each test.

Figure 6 displays the average results of five test runs with GHOST version 3.0. According to these results, the hload varied by very little when the polygon count increased for a single object. However, when polygon counts exceeded about 120k, GHOST started to produce random errors, even though hload was under 0.2ms. From these results, for a single object with no overlapping polygons, we found the GLB and LUB to be 120k and 235k polygons respectively.

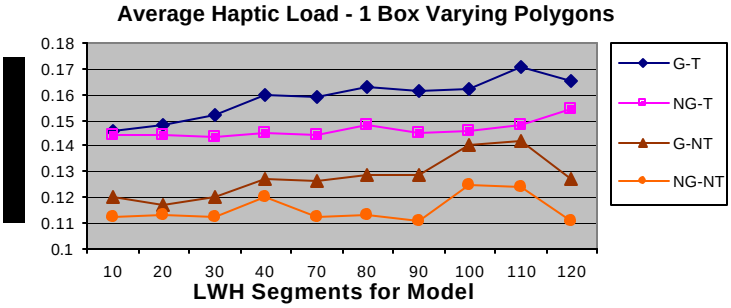


Figure 6: Avg test results for increasing polygons for one object

For multiple polymesh objects, the number of polygons is kept consistent with each increment step in the previous test. Multiple 10x10x10 segment boxes, each containing 1,200 polygons, are used to form the polygon count for each test run. The vertex count was greater in the multiple polymesh objects test than in the single polymesh object test by 2(N-1) vertices, where N is the number of boxes. The increase in hload is relatively linear as objects are added to the scene, Figure 7. Further testing gave the GLB and LUB to be 35 objects (42k polygons) and 49 objects (58.8k polygons) for these polymesh objects with non-overlapping polygons. The limit of 1ms was exceeded for 49 objects, therefore not allowing for the true value to be recorded. However, we did notice that with only 35 objects, the

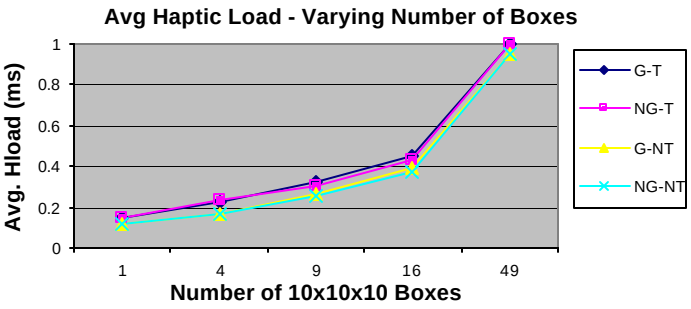


Figure 7: Avg test results for increasing number of objects

number of polygons from the single polymesh object test was reduced from 120k to 42k in the multiple polymesh objects test. Even with an additional 78k polygons the maximum hload was about 80% less.

We estimate that on the average, collision detection for the polymesh objects is about 71-77%, while graphics is about 2-8%, and touching the object is about 19% of the hload. Tests also showed that the graphics increased hload dependent of the number of polygons. Touching the polmesh, however, raised hload on average by about 26%.

4. Conclusion

In this paper, we addressed scene complexity issues based on the maximum number of primitive objects as well as the maximum number of polygons for single and multiple polymesh objects that allow stable haptic interactions when using GHOST. We discussed the idea of specifying a scene complexity stability range. This range was defined by the greatest lower bound (GLB) and the least upper bound (LUB). GLB is the highest scene complexity for which the application always runs with stability and produces no errors. The LUB is a point at which the haptic application will no longer run and produces errors instantly after haptics is initialized. When the scene complexity is between GLB and LUB the application runs, but is unstable and prone to errors. We were also able to estimate the percentage of hload used for the collision detection, graphics, and user touching an object.

Though these initial tests gave us complexity bounds for some haptic applications, more tests need to be performed in order to estimate the feasibility and performance expectations for a general haptic environment that involves many other levels of complexity. For example, other tests results indicate that hload increases if the point of contact is within multiple bounding boxes and increases even more if the point is touching multiple objects. Tests also show that there can be different hloads within a single polymesh object, depending on its topology (eg. corner points, overlapping polygons, etc.). Understanding scene complexity limits plays a key role in creation of haptic applications by providing critical data needed to estimate the feasibility and performance for generic haptic environments.

5. References

[1] Ho, C., Basdogan, C., Srinivasan, M.A., 1999, "An Efficient Haptic Rendering Technique for Displaying 3D Polyhedral Objects and Their Surface Details in Virtual Environments", October 1999 Vol. 8, No. 5, pp. 477-491, Presence: Teleoperators and Virtual Environments.

[2] Acosta, Eric J., Haptic Virtual Environment, M.S. Thesis, Computer Science Department, Texas Tech University, May 2001.

[3] Farida Vahora, Bharti Temkin, Thomas M. Krummel, Paul J. Gorman, "Development of Real-Time Virtual Reality Haptic Application: Real-Time Issues", *12th IEEE Symposium on Computer-Based Medical Systems* - CBMS 1999, June 18-20, pages 290-295, ISBN 0–7695–0234–2

[4] GHOST Software Developers Toolkit Programmers Guide, SensAble Technologies, Inc.

The Sound and Touch of Mathematics: a Prototype System

Frances L. Van Scoy, Takamitsu Kawai, Angela Fullmer, Kevin Stamper
West Virginia University
Iwona Wojciechowska - Alderson Broaddus College
Addis Perez, Jesir Vargas - University of Puerto Rico-Rio Piedras
Shelma Martinez - University of Puerto Rico-Mayaguez

The West Virginia Virtual Environments Laboratory is studying ways to use virtual environments technology to assist users with various disabilities. One problem of special interest to us is how to teach mathematics to students with vision disabilities. We are using the PHANToM and sonification techniques to display mathematical functions to visually impaired students.

Rationale

We agree with Moses and Cobb, "Part of the literacy standard, then, the floor for all students, must be this: when you leave middle school, you are ready to engage with the college preparatory sequence in high school. It's a moving target, but however it's defined, it must then be seen as another floor: when you leave high school, you must be able to engage college curricula in math and science, for full college credit." [1]

National Science Foundation Director Rita Colwell has said, "Every schoolchild must be educated for a productive and contributory place in an advanced information age... K through 12 is the real challenge. As a start, we begin with the assumption that all children can be educated in math and science. This may sound so elementary as to be downright silly! In some places, the educational approach is to sift and sort students early-on. This tells some students right at the starting gate that they can't master science and math -- that we do not expect them to succeed. This becomes a self-fulfilling prophecy, damning to the student and destructive for the country. We must believe in all children so that they learn to believe in themselves..." [2]

Our goal in this project is to help students with disabilities or with different learning styles learn pre-calculus so that college majors in science and engineering are available to them.

Previous Work

The current project brings together previous work in two areas, use of the PHANToM in displaying map information for pre-trip planning by those with visual impairments [3] and sonification of complex data sets [4-7]. Our sonification work emphasizes the development of algorithms for composing music from complex data sets which will be pleasant to listen to and will assist listeners in discovering data relationships which are not otherwise obvious.

In 2000 we developed a program for the PHANToM which accepts as input a mathematical function of one variable and then constructs a haptic model of a solid block on whose front face a groove representing the function has been carved. The user types on the computer keyboard to enter commands such as those for changing bounds of the domain and range, and the program uses spoken feedback to communicate with the user. A important component of our system is a compiler written using flex and bison which parses the user's function written in Fortran notation [8].

Stephen Brewster's group at the University of Glasgow has built a prototype multimodal haptic math system and done human factors research with both blind and sighted users. In the 2000 version of their system functions were provided by the human factors researchers and hard-coded.

They tested for effectiveness of different values of friction and representation of grid lines. They referenced their own work in sonification, writing, "An effective way of presenting the overview of the line graph will shorten the time required in this process and give blind users a better understanding about the graph. Using non-speech sound to provide this kind of quick overview [is] being investigated." [9]

Specification of New Prototypes

In 2001 we extended our prototype haptic math system to add:

(1) improvement of the user interface,

(2) display of functions of two variables, and

(3) sonification of functions of one variable

Improving User Interface

The existing version of haptic math provided only the basic functions needed to allow the user to enter a function and change some parameters. All input was done via the keyboard and made several assumptions to simplify the initial interface. Work this summer focused on identifying components required and/or useful in a more complete user interface. This includes not only the listing of these components but consideration as to how best to implement each of them to allow the greatest adaptability for users with varying disabilities or learning styles.

There are two main parts of the user interface: the informational display and the system controls.

The display part of the interface is concerned with how the function is represented to the user. This includes the graphic representation as drawn on the screen, the haptic representation as felt via the PHANToM, and the production of any auditory information. In regards to the multi-modal 'display' there were several issues considered. These included investigation into the way that the curve is drawn to provide a more uniform groove width at points with steep or zero slope, methods of informing the user (other than visually) of discontinuities in the graphed function and providing the user with alternative methods to explore the graph.

The system control part of the interface is concerned with how the user sets parameters, enters functions and in general conveys information to the system. For the system control part some issues considered include variations in the format of the functions that may be entered, the means of detecting and informing the user of errors in the function entered, the provision of user access to parameters and the ability of the user to selectively alter parts of the multi-modal display to adapt to individual needs. The existing version restricts the choices for system control keys to those on the left hand. This is an unnecessary and potentially problematic restriction for several reasons. It assumes a right-handed user. It assumes the traditional 'qwerty' keyboard, a problem since other keyboards are used by potential users. Also, it forces some command sequences to be rather arbitrary. The restriction is unnecessary as it is not possible on a qwerty keyboard to use only left hand sequences to enter characters such as the parentheses needed in the functions entered. This also forces the user to memorize the correspondence between keys and parameters - a potential problem for a novice user who may not even realize what parameters are available. It would also be desirable for the user to be able to alter the display. For example, a user might find the musical representation of the curve distracting and choose to turn it off.

Display of Functions of Two Variables

We also developed a new module for displaying functions of two variables.

First, the visual representation had to be changed. The original one-variable version carves a groove in a block of wood, giving a 3-d representation of a 2-d curve. In the case of a function of two variables, we are dealing with a 3-d surface. To obtain this surface, the plane (x,y) is triangulated and the z-value of the function is calculated for each triangulation vertex. This produces the set of 3-d points (x,y,x) that are used to create the tripoly mesh that PHANToM user can "feel." To make the surface easier to explore without falling off the surface, we added a bounding box.

The second change involved modifying the function parser to accommodate functions of two variables. This major modification required making changes to the interface between the display module and the parser. Additional arguments were needed. Also the type and meaning of one of the parameters was changed. The latter change was necessary, because of the precautions taken in the previous version against mathematical roundup errors.

Work remains to be done. Currently the function surface is displayed with the (x,y) plane horizontal. Additional display options can be added such as displaying the plane vertically or allowing the user to rotate the surface. Also for sighted users the color of the display should be changed and lighting added to create shadows. An artifact of the 2000 system is that the expression defining the function is re-parsed for every pair (x,y), with a major loss of efficiency. We need to restructure the system so that the function is parsed one time and then evaluated once for each (x,y) pair.

Sonification of Functions of One Variable

In sonifying functions of one variable we map a function to a line of music, mapping x-coordinates to time and y-coordinates to pitch. We do this in order to give the student a way of remembering the shape of a function in the belief that many students will be better able to remember a melody describing the sine function rather than only the muscular sensations of tracing it. We use two different encodings--one using the western chromatic scale and one using micropitch--in the belief that melodies based on the familiar scale will be easier to remember but that micropitch will give a more accurate representation of a function (since the chromatic scale encoding essentially represents continuous functions as step functions).

Figure 1 shows an example illustrating how we sonify functions of one variable. The music is that generated for f(x) = tan(x), for 0 = x = 2 p.

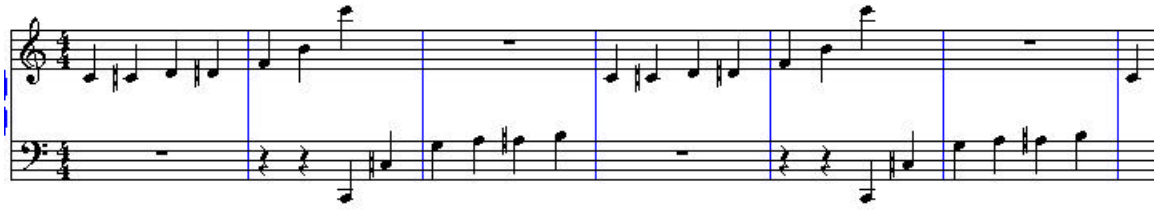


Figure 1. Sonification of Tangent Function

Continuing Work

We are now designing a human factors study for testing this PHANToM-based prototype with secondary school students and doing preliminary design for some course modules incorporating this approach.

References

[1] Moses, Robert P. and Charles E. Cobb, *Radical equations: math literacy and civil rights*, 2001, Boston: Beacon Press, p. 16.

[2] Colwell, Rita, address to DC Science Writers Association, September 8, 1998, available at http://www.nsf.gov/od/lpa/forum/colwell/rc80908.htm

[3] Van Scoy, Frances L. Vic Baker, Chaim Gingold, Eric Martino, Darrin Burton, "Mobility Training using a Haptic Interface: Initial Plans," in Salisbury, JK and Srinivasan, MA (Eds), "Proceedings of the Fourth PHANTOM Users Group Workshop," AI Lab Technical Report No. 1675 and RLE Technical Report No. 633, MIT, November 1999, available at http://www.sensable.com/haptics/community/pdf/PUG1999.PDF

[4] Van Scoy, Frances L., "Sonification of Complex Data Sets: An Example from Basketball," Proceedings of VSMM99 (Virtual Systems and MultiMedia), Dundee, Scotland, September 1-3, 1999.

[5] Van Scoy, Frances L., "Sonification of Remote Sensing Data: Initial Experiment," Information Visualisation 2000, London, UK, July 19-21, 2000

[6] Mutnuri, Sunitha, "Sonification of GIS Data," Master of Science in Computer Science problem report, Lane Department of Computer Science and Electrical Engineering, West Virginia University, December 2000.

[7] Wallace, Carl, "Sonification of Basketball Data," Master of Science in Computer Science problem report, Lane Department of Computer Science and Electrical Engineering, West Virginia University, December 2001.

[8] Van Scoy , Frances L., Takamitsu Kawai, Marjorie Darrah, Connie Darrah, "Haptic Display of Mathematical Functions for Teaching Students with Vision Disabilities: Design and Proof of Concept," In *Proceedings of the First Workshop on Haptic Human-Computer Interaction*, Glasgow, Scotland, August 31-September 1, 2000, available at http://www.dcs.gla.ac.uk/%7Estephen/workshops/haptic/papers/haptics_and_hci_workshop.zip

[9] Yu, W., R. Ramloll, and S. A. Brewster (2000). "Haptic graphs for blind computer users," in *Proceedings of the First Workshop on Haptic Human-Computer Interaction*, Glasgow, Scotland, August 31-September 1, 2000, available at http://www.dcs.gla.ac.uk/~stephen/papers/Assets2000.pdf

Acknowledgements

This work is being conducted in the West Virginia Virtual Environments Laboratory in the Lane Department of Computer Science and Electrical Engineering of West Virginia University and is supported by the EPSCoR programs of the National Science Foundation and the state of West Virginia and by the NSF CISE Research Experiences for Undergraduates program.

Darrin Burton of the WVU Center for Assistive Technology has provided valuable advice on usability of the system.

Nicole Johnson, a WVU McNair scholar, and Yusuke Abe, a Fairmont State College student, were active participants in VEL discussions about this project.

A Dynamic Design Strategy for Visual and Haptic Development

Arthurine Breckenridge
Interaction Laboratory[1]
Sandia National Laboratories
arbreck@sandia.gov

Ben Hamlet
Sandia National Laboratories
brhamle@sandia.gov

Derek Mehlhorn
University of Washington
dtmehlh@sandia.gov

Kevin Oishi
Carnegie Mellon University
ktoishi@sandia.gov

Abstract

The evolution of modern computer programming languages comes with the need for the strategies with which we implement them to change as well. Fully dynamic and reusable visual and haptic simulations are now possible given the modular nature of current programming languages. The new standards for simulations are dynamic applications that can load and utilize code modules without shutting down, and that will almost never require a complete rebuild for new applications. Two major issues we addressed and implemented in pursuit of this standard are:

I. Managing the resources of one machine
II. Dynamic Human Computer Interface (HCI)

This paper discusses the limitations of current development strategies, and presents the work done at Sandia National Laboratories to develop an application that will conform to the new dynamic standards.

1. Limitations of Current Simulation Strategies

Current Virtual Reality (VR) simulation strategies have two major limitations. First, they do not exceed the resources of one machine. The current strategy for VR simulations of all kinds is to create a specifically tailored application, from the ground up, to accomplish a specific goal. Not only are these applications designed to simulate only one situation or event, but, once completed, they cannot be updated or modified without being completely recompiled. There are two main drawbacks to this development strategy. The applications, or parts of them, are difficult to reuse in future work, and the future simulations must be almost completely re-written. A number of attempts have been made to reduce the need for complete program rewrites. These come in the form of Application Programming Interfaces (API's).

Second, current VR strategies do not support dynamic Human Computer Interaction (HCI). HCI has traditionally been limited to 2-dimensional devices like mice or keyboards. When working with a 3-Dimensional simulation or programming environment, working with a 2-dimensional input device is counter intuitive and limiting to the user. Development in the field of computer haptics has improved HCI in a

[1] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

[2] PHANToM is a registered trademark of SensAble Technologies, Inc.

number of ways, including a much more intuitive virtual environment. Computer haptics has great potential in the VR field, not only for simulation purposes but for simulation development [1] as well. Such devices as SensAble Technologies' Desktop PHANToM[2] [2] provide a user with 6 degrees of freedom of motion (x, y, z, yaw, pitch, and roll) as well as 3 degrees of force feedback (x, y, z). Not only do haptics devices, such as the PHANToM, allow users to interact with a computer in 3-dimensions, but they also allow the computer to interact with the user physically, in addition to the standard visual feedback. These devices, being developed into new APIs, have proven to be intuitive and efficient in the development and interaction with virtual environments [1,3]. The improvement of HCI and the emergence of API's have increased programming productivity and reusability, proving a more efficient and desirable design strategy is becoming possible.

2. A Dynamic Standard

In the past, simulation development has been limited by the linear nature of early programming languages, such as Fortran and Cobalt. Newer languages, such as C++ and Java, are now object oriented, encouraging modular code development, and support for greater code reusability. The modular nature of current programming languages lends itself well to a new strategy for VR and application design.

One of the limitations, needing to be addressed, of current VR strategies is the lack of reusability. Extremely powerful and impressive applications and simulations have been developed as single use applications. Therefore, these program's contributions to their fields are static. They cannot be easily modified to perform different tasks, nor can they be easily disassembled and applied directly to another application. Current computer hardware and software technology make possible a number of improvements to VR development strategies, including the ability to manage the resources of a machine through dynamic loading.

Simulations can now be designed in a modular nature that allows for dynamic scalability and reusability. The new standard for VR development should be multiple use applications that can dynamically load new features without recompilation. Current API's should be designed in this manner to increase productivity and reusability. The primary benefit of a new dynamic standard is the reusability and expandability that will allow for more robust and extensive simulations than ever before. Dynamically loadable modules can also enable the simulations to be run on a much larger range of machines, since users can dynamically customize the resolution, or complexity, of the application before or during execution. People using the simulation will only load the modules they need or can support with their machine. This strategy will also enable a large number of people to develop VR simulations without being programming or design experts.

The second improvement to current VR development strategies that we worked with lies in HCI. One limitation of current HCI is that, for the most part, it is limited to a single user with a static user interface. Currently, the means by which users interact with their virtual environments do not change with data needs. Therefore, in order to maximize the breadth of an application, all conceivable features are loaded upon execution, a method that will quickly exceed system resources. Our implementation of a dynamic HCI includes the ability to dynamically change the way that one can interact with one's simulation environment by using a series of dynamic menus that can be developed during runtime. The ability for multiple users to work collaboratively over a network in a Virtual Collaborative Environment (VCE) would also be extremely valuable, and has limitless potential. Productivity and quality of products will be increased as designers and customers collaboratively design in a VCE, even if they are hundreds of miles apart.

3. An Implementation Example

3.1 Overview

An initial attempt to implement the new standards of VR development, as stated above, has commenced at Sandia National Laboratories' Interaction Lab. The basic methods and strategies used are applicable to more general cases of VR development. It should be noted that the machines used in development were little more than commercial-off-the-shelf PC systems, readily available to most individuals interested in the new wave of VR development. Novint Technologies' e-Touch[3] was used as a basis for haptics development with Sensable Technologies' Desktop PHANToM

[3] e-Touch is a registered trademark of Novint Technologies, Inc. http://www.etouch3d.org

device. Novint is an internet spin-off company, from the Sandia National Laboratories' Interaction Lab. e-Touch is an open module effort to provide a 3D application programming interface as well as a graphic/haptic user interface (GHUI) to the internet community.

The project is tailored towards an architectural design application. However, since the project is meant to implement the new standards for haptics and visual simulations, the application area goes far beyond a simple CAD program. At the core of the project is the need to manage the resources of a machine, to dynamically add features, ranging from rendered objects to new modules of code, without having to exit the application. Other goals include improving our HCI by supporting multiple users, working collaboratively across a network, using dynamically changing tools and interfaces.

objects are designed to be independent code modules that can be dynamically loaded upon request. Commands are code modules that are used to interface objects and tools with the foundation of the application.

To support expandability and dynamic loading of different features, a series of base or template classes exist within the foundation of the application. New objects, tools, and commands can then be derived from these classes, providing expandability and compatibility. This structure also enables novice users to implement basic features and provides advanced users with the ability to overload the base implementation and create unique and complex additions to the simulation. Objects and tools are also designed in a way that they depend only on the lowest level of our API, Novint's e-Touch. Command modules are used to interface them with specific applications. Therefore, the same objects and tools

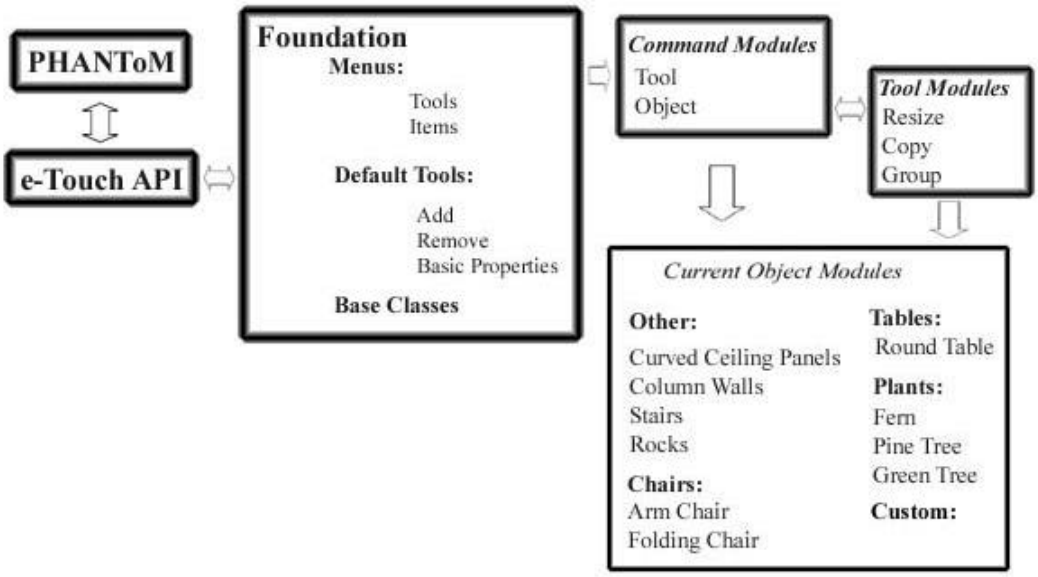


Chart 1. **Shows the design structure of the application.**

3.2 Basic Structure

The application structure breaks down into three major categories: objects, tools, and commands. Objects are graphic and/or haptic items that can be dynamically added, removed, and manipulated within the simulation. Tools enable a user to interact with one's environment and with the objects therein. The tools, except for a select few default tools, and all

can be reused in an infinite array of e-Touch based applications by simply swapping out their command modules.

Chart 1 shows the basic structure of our application. The foundation classes consist of, essentially, the groundwork for objects, menus, and tools. An object is defined as anything rendered with either graphics, haptics, or both. For our purposes in the architectural application, objects are

created to use both styles of rendering. Tools are usually represented by a 3D cursor, which is directly controlled by the PHANToM for input. Tools are used to alter the state of the application by adding, copying, deleting, and in other ways manipulating an object. Tools generally manipulate or change some property of an object. Menus are completely three dimensional, and contain 3D objects, such as buttons, sliders, and knobs. An explanation of the menuing system can be found in section *3.4 Menus.*

At this point, it should be noted that this project's system of combining graphical and haptic representations of objects did not require the redevelopment of 3D modeling applications. The aim is not to replace existing, proven, and refined applications, but rather to address a new method of efficient development that can use and improve upon existing programs. The graphical nature of objects is still designed using more specific, existing, applications. The project currently uses imported 3D Studio Max[4] files to create the graphics for our haptic objects, and contains a framework for converting other types of 3D object files into data that can be used by the application. Using these pre-designed graphics, a haptic representation is either specifically defined or generally applied based on a series of parameters, yielding a completed haptic object.

An important concern with any system that combines haptics and graphics data is the need for an on disk storage system that also ties these two representations together. This can be accomplished through the use of a common file format that stores both object depictions in a single file. Such a format would allow even the most novice users to create simple objects, while more advanced and knowledgeable users would still be able to create the complex, individualized, and ornate objects they require.

3.3 Tools

Our application provides a dynamic and 3D GHUI. A series of default tools allow users to interact with their environment instantly in several basic ways, while a series of 3D menus dynamically reconfigure themselves to incorporate new tools during runtime. *Image 1* shows an object being resized using one of the applications default tools. *Image 2* shows the default property manipulation tool. Without any additions to the applications, default tools enable users to add, delete, resize, and manipulate object properties, while still possessing the ability for custom tools to be loaded in at anytime for customized interaction. This system increases productivity, decreases lost time, and allows for a more robust and overall user-friendly application. The range of properties that can be manipulated is also dynamic, allowing users to specify unique properties that they wish to change for only select types of

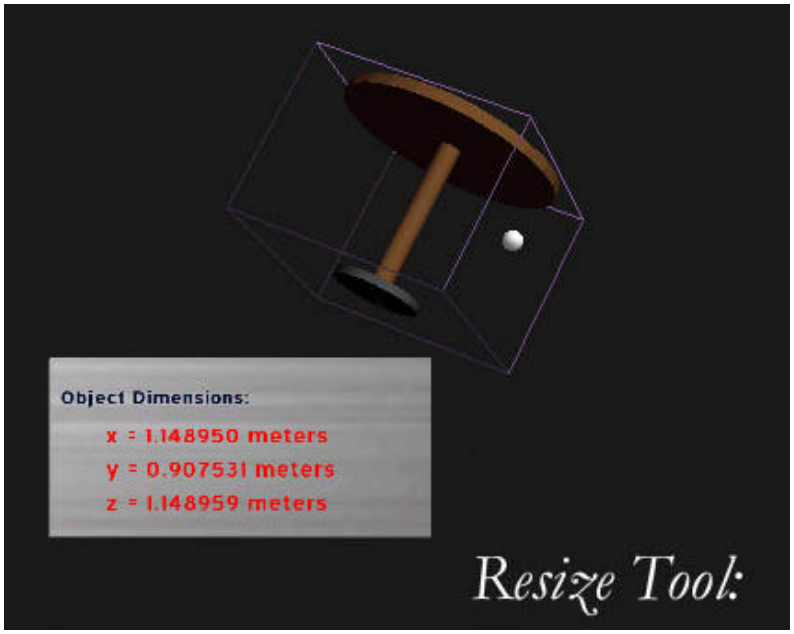


Image 1. Shows an object being dynamically resized using the default resize tool.

[4] 3D Studio Max is a registered trademark of Autodesk, Inc.

objects. Currently, the application enables manipulation of both the size and color scheme of an object while the program is running.

In a system that allows for real-time extendibility, it is necessary for applications to make use of dynamic tools, and dynamic menus to interface with these tools. It is imperative for various methods of interacting with the virtual environment to be added as required. This interaction should also be able to take place in a way never imagined by the initial developers of the system, and should not be limited to any basic, underlying, and restricting subsystem.

A method for completing such a task has been implemented for this project. Several tools have been added without restructuring or altering the foundation of the application. An example of this is a grouping tool, which allows users to group together any number of objects. Then, any change made to an object in a group is applied to all other objects in that same group. Such tools are derived off a base tool class, which contains only those properties common to all tools, and a means to interface with e-Touch. They are independent of the foundation application, and are only specific to the very general object type that is used throughout the application. This proves the ability of the application, but more importantly, the new standard of development, to allow for the kind of extendibility warranted and required by such an ambitious project.

3.4 Menus

A menu system is another aspect of this project. An important part of this feature is the three dimensional nature of the menus, whose basic implementation is in the e-Touch API. Although visually similar to older windowing systems, giving them a sense of familiarity, these menus are quickly realized to be much more functional than their 2D counterparts. Menu objects, such as buttons and sliders, are also in 3D. Using the PHANToM haptic device, sliders are moved, buttons are clicked, or knobs are turned to adjust aspects of an object. This is a very powerful feature when coupled with a device such as the PHANToM, as buttons and other objects have a physical response when they are activated. Users can feel 3D buttons being pushed, similar to their real world equivalents, such as the buttons on a telephone. A particularly useful example is the 3D slider used for changing the color of an object. A wire frame cube is used to represent all available colors. The slider can be positioned by the user at any point in the cube, and the object's color is determined by the slider's location, see *Image 2.*

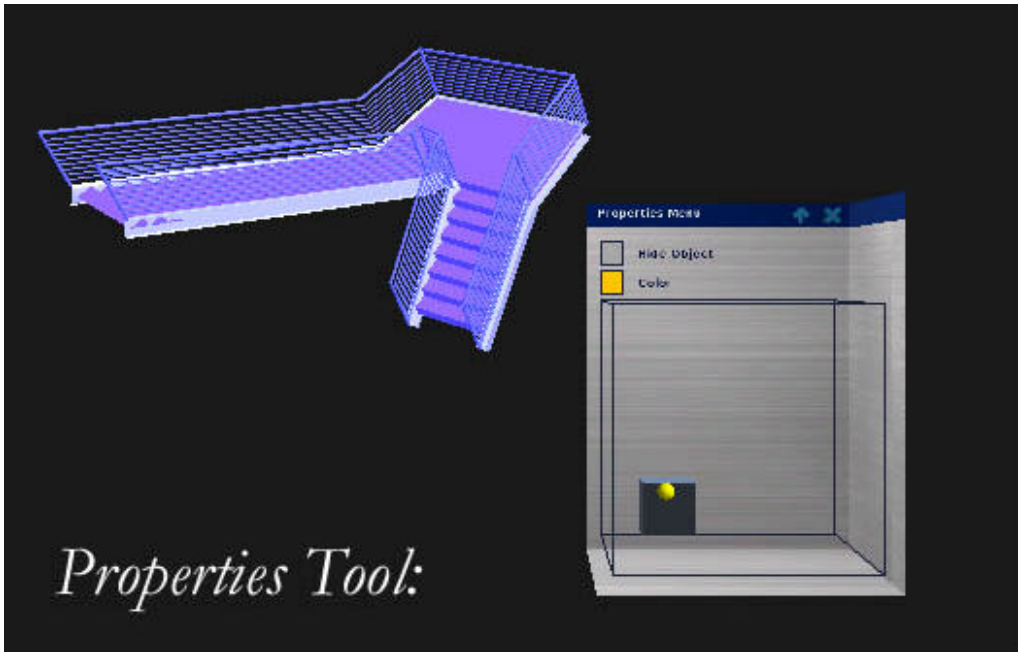


Image 2. **Shows an object's color property being manipulated using the default properties tool and it's corresponding dynamic menu.**

As our particular application strives to exceed the resources of a single machine, which basically means it needs to be expandable at runtime, a dynamic menuing system was built on top of a set of default, foundation menus. An application area where users are able to easily add a variety of objects to the simulation is needed. This is accomplished by way of a menu system that recognizes the addition of new objects and enables their use. Specifically, a particular folder in the application directory has been set aside for storage of objects the program may use. When a user decides to add an object, this directory is scanned for subfolders, which are presented as buttons on an object adding toolbar. When a button on this toolbar is pressed, a corresponding directory is scanned, and, dynamically, a new object selection menu is built. Users then choose which object they wish to add by pressing any of the buttons on the menu. The menu is destroyed when closed, allowing for new source files for objects to be added to the program directory at runtime. These objects will then be represented in the menus the next time an object is to be added. The menuing system is depicted in *Image 3*. Currently, the project has a dynamic menu that recognizes the addition of new objects using a general haptic object file format and allows the user to use new objects without halting program execution. The goal of dynamic loading without shutting down is aimed toward being able to access features from the internet dynamically.

3.5 Lessons Learned

Developing an application that conforms to the new standard for visual and haptic development has not been a small task. Although modular programming languages allow programmers to write code that maintains a structure that can support dynamic loading, there is not currently a good program that allows for dynamic loading of code modules. The most promising is a program called Bamboo [6]. Other options include placing code modules within dll's (dynamic linking libraries). This enables both dynamic linking, adding code at the start of execution, and dynamic loading, adding code during execution.

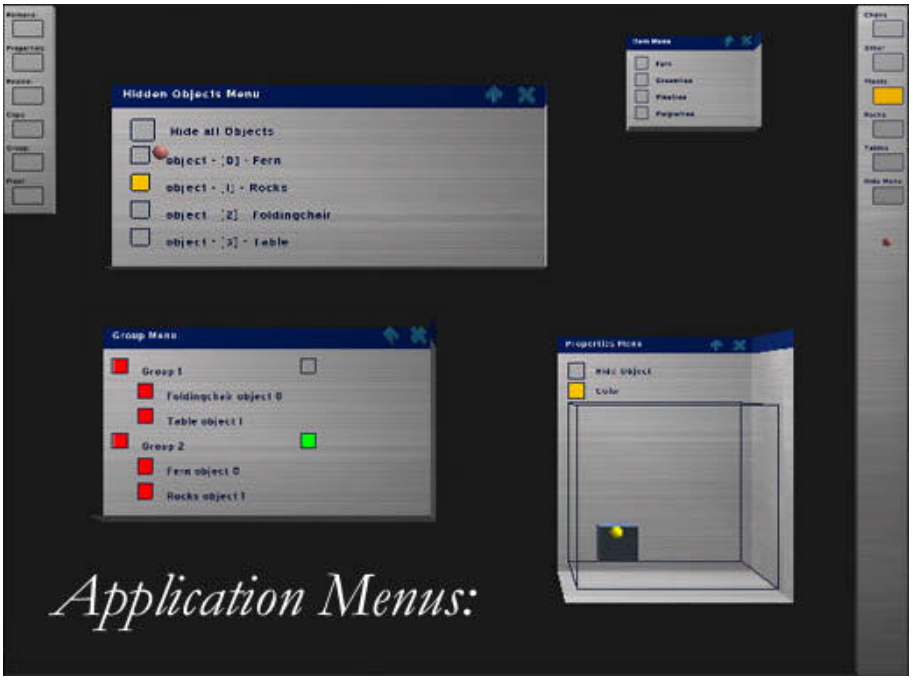


Image 3. **Shows a number of dynamic menus that make up our unique HCI.**

In order to improve current HCI, we were working with SensAble Technologies' Desktop PHANToM. During our work we discovered several limitations of the current haptic technology. For example, a limited range of motion potentially requires scaling of the

device motion. Other recognized haptics needs involve a standard format for object property definition. Since graphics and haptics are produced by different output devices, a monitor versus a PHANToM, they have always been treated as two separate entities. Creating a standard format for haptic object information is a necessary step in the development and continuation of the computer haptics field. A behavior constraint library that can be referenced and used to define behaviors and characteristics of specific objects is one promising approach to a common file format. Investigation and communication with the haptics community at large is required for this format to be as inclusive and useful as possible.

Issues to consider regarding supporting multiple users to improve and develop HCI include, but are not limited to, network latency and the effects it has on force feedback[5,6], and platform independence. Haptics is difficult to successfully perform over a network due to the high refresh rates and the subsequent and inherent sensitivity of such simulations. Also, because of the complicated nature of haptics, threads are inexorably an integral part of any simulation. Due to the often platform specific nature of threads, this is a complex and inevitable problem encountered with networking individuals using various system configurations. In our goal for reusable and modular code, platform independence is an important obstacle to overcome.

4. Conclusions

Given the evolution of programming languages and techniques, it is possible for more efficient and productive VR applications to be designed. It is now possible to exceed the resources of one's machine and to improve and develop current HCI methods. Specifically, multiple use, multiple user simulations that are composed of dynamically loadable modules, have become a reality and present a bright future for VR development and expansion.

An architectural design application based on and implementing the concepts and ideas presented in this paper is currently under development at Sandia National Laboratories. The foundation of the application has been completed utilizing Novint Technologies' haptics API e-Touch. We have been successful in importing various unique objects from 3D Studio Max files into our simulation, and are able to manipulate their location, orientation, and physical properties dynamically with a number of tools that have been written into the application foundation. The modular, and eventually dynamically loadable, structure of our application has also been verified using several independent tool modules that successfully interact with objects inside the simulation.

The work at Sandia National Laboratories already shows great potential for the full implementation of the ambitious VR standards set forth in this paper. Dynamically expandable, multiple user simulations with a high degree of reusability are on the threshold of reality, and are the future of VR design and development.

5. References

[1] Anderson, Tom, "FLIGHT: A 3D Human-Computer Interface and Application Development Environment", Proceedings of the Second PHANTOM Users Group Workshop, Cambridge, MA, 1997

[2] SensAble Devices Inc., "The PHANToM" literature from SensAble Devices Inc. 225 Court St., Vanceburg, KY 41179.

[3] Gutierrez T., Barbero J.I., Aizpitarte M., Carrillo A. R., and Eguidazu A., "Assembly Simulation Through Haptic Virtual Prototypes", Proceedings of the Third PHANTOM Users Group Workshop, Cambridge, MA, 1998

[4] Matsumoto S., Fukuda I., Morino H., Hikichi K., Sezaki K., Yasua Y., "The Influences of Network Issues on Haptic Collaboration in Shared Virtual Environments", Proceedings of the Fifth PHANTOM Users Group Workshop, Cambridge, MA, 2000

[5] Hespanha J., McLaughlin M., Sukhatme G., Akbarian M., Garg R., Zhu W., "Haptic Collaboration over the Internet", Proceedings of the Fifth PHANTOM Users Group Workshop, Cambridge, MA, 2000

[6] Watsen K., Zyda M., "Bamboo- a portable system for dynamically extensible, real-time, networked, virtual environments", Virtual Reality Annual International Symposium. Proceedings, Atlanta, GA, 1998